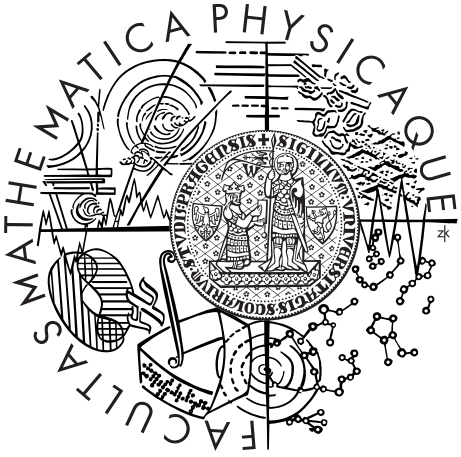
Charles University in Prague
Faculty of Mathematics and Physics

MASTER THESIS



Pavel Nohejl

Experiments with Linked Data

Department of Software Engineering

Supervisor of the master thesis: Mgr. Martin Nečaský, Ph.D.
Study programme: Computer Science
Specialization: Software Systems

Prague 2011

Na tomto místě bych rád poděkoval Mgr. Martinu Nečaskému, Ph.D za konzultace a cenné rady při psaní této práce. Velký dík patří i mé rodině za dlouhodobou podporu při studiu.

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V Praze dne 5.8.2011 Pavel Nohejl

Název práce: Experiments with Linked Data
Autor: Pavel Nohejl
Katedra: Katedra softwarového inženýrství MFF UK
Vedoucí diplomové práce: Mgr. Martin Nečaský, Ph.D, Katedra softwarového inženýrství

Abstrakt: Tato diplomová práce si klade za cíl vytvořit „manuál“ k technologii Linked Data. První část práce stručně popisuje Sémantický Web a jeho vztah k Linked Data, které jsou poté podrobně vysvětleny. Vysvětleno je jak vytvářet Linked Data dle takzvaných „Linked Data principů“ včetně potřebných technologií a nástrojů. Druhá část práce obsahuje praktické zkušenosti s vytvářením a užíváním Linked Data. Podrobně je popsáno získávání dat o veřejných zakázkách pomocí pro tyto účely vytvořeného web crawleru. Na to navazuje popis transformace těchto (relačních) dat do formátu Linked Data včetně jejich propojení s ostatními Linked Data zdroji. Součástí práce je také vytvoření aplikace pracující s Linked Data, což porovnáváme s klasickým přístupem, kdy aplikace čerpá data z relační databáze. Toto porovnání je doplněno benchmarkem. Nakonec je uveden návod pro začínající vývojáře shrnující naše zkušenosti a poznatky včetně problémů, které je dle našeho názoru třeba vyřešit, aby se Linked Data jako základ Sémantického Webu mohli dále rozšiřovat.

Klíčová slova: Linked Data, SPARQL, RDF, Sémantický Web

Title: Experiments with Linked Data
Author: Pavel Nohejl
Department: Department of Software Engineering
Supervisor of the master thesis: Mgr. Martin Nečaský, Ph.D, Department of Software Engineering

Abstract: The goal of this master thesis is to create a “manual” to Linked Data technology. The first part of this thesis describes the Semantic Web and its relationship to Linked Data. Then follows a detailed explanation of Linked Data and so called “Linked Data principles” including involved technologies and tools. The second part of the thesis contains practical experiences with creation and using Linked Data. Firstly is described obtaining data on public procurement by web crawler developed for these purposes, followed by a description of transformation obtained (relational) data into Linked Data and their interlinking with external Linked Data sources. One part of this thesis is also an application consuming created Linked Data. This is compared with the traditional approach when the application consumes data from a relational database. This comparison is supplemented by a benchmark. Finally is presented a manual for the beginning developer which summarizes our experiences. The list of problems which are necessary to solve (from our point of view) for further development of Linked Data is also included.

Keywords: Linked Data, SPARQL, RDF, Semantic Web

Contents

Chapter 1

Introduction

The growth rate of the Internet exceeds that of any previous technology. Measured by users and bandwidth, the Internet has been growing at a rapid rate since its conception, on a geometric or even exponential curve [1].

On today's Internet it is possible to find almost everything we can imagine. But as the amount of documents, data and web pages grows it is getting harder and harder to find relevant, correct and trustworthy information.

Search engines help us to find documents containing specific words, but the results of searching are often inaccurate and the content is not exactly what we wanted. Searching is based on the web page content instead of its semantic meaning or information about the page. Web pages are presented in natural language and contain only information about how their content should look like (size, paragraph spacing, etc.), but no information about the semantic meaning. Links between web pages have no information about semantic relationship between linked web pages. Furthermore if web-developers want to enhance their Web applications, they must use proprietary APIs (Google, Facebook, etc.) designed in different ways and using proprietary data formats. Web-developers must learn new interfaces over and over again.

"The Web of Data, also known as the Semantic Web, has promised for a long time to resolve these issues. To date, however, only partial solutions to real-world problems exist, many of them addressing rather toy datasets — that is, small, artificial, non-real-world, data sets that do not demonstrate scalability for the Web." [2].

Since Tim-Berners Lee has proposed his idea of Linked Data as a way to publish data online in machine readable form the situation changed dramatically. The amount of datasets published according to so-called "Linked Data principles" has started to grow same as applications consuming these data or tools helping with Linked Data creation. DBPedia, literally the center of Linked Data world, has started to offer structured machine-readable data extracted from Wikipedia

and became the biggest interlinking hub. Tools for mapping data from relational databases to Linked Data has started to appear and develop. Governments have begun to promote the publication of data according to the Linked Data principles.

The Chicken-and-Egg Problem: no data because of no applications or no applications because of no data, is gone. Applications, tools and data are available. Nothing prevents us from building the Web of Data.

1.1 Background and motivation

There exist already a lot of tools for transformation of data into Linked Data, their mapping from relational databases or interlinking. There exist also a lot of Linked Data sets with SPARQL endpoints or applications consuming these data. However, a lot of mentioned tools, datasets or applications are still in an experimental phase or under active research. Most of them are intended to be only used in academic sphere. They do not have a user friendly GUI (graphic user inferface). It is necessary to download manually additional libraries and to do complicated initial configurations. A developer of current web applications uses mature technologies, platforms, programming languages and frameworks. Tools involved in Linked Data creation have been vastly improved in the past several years. Nevertheless it could be still a burdensome task to find appropriate tools which pass the most developer's needs in order to transform an ordinary dataset into Linked Data.

The aim of this thesis is to help developers who decided to experiment with Linked Data. We will try to identify if use of Linked Data is beneficial for ordinary web developer and if the effort of Linked Data creation is worth the benefits it will bring. The thesis will provide an overview of tools involved in Linked Data creation or transformation out of ordinary dataset and discuss different steps and difficulties it can bring. As a lot of data are backed by relational databases the thesis will also focus on transformation of relational data into Linked data. The thesis will minutely describe our own experiences with implementing web crawler scraping governmental data, creating a Linked Data set out of existing (relational) dataset and building an application consuming Linked Data. Finally a benchmark of our application will be presented.

1.2 Structure of this document

The second chapter briefly introduces the Semantic Web and technologies on which depends. The next chapter describes Linked Data, its principles and steps for transforming a dataset into Linked Data, involved technologies like RDF or

SPARQL query Language. Existing Linked Data datasets and application are introduced. Then following chapter gives an overview of the tools involved in different steps of Linked Data creation. The tools for mapping relational data to RDF, interlinking or publishing Linked data on the Web are presented in the same way as the tools for their storage. Chapters 5 and 6 present our experiences and difficulties we had during transforming ordinary dataset to Linked Data and development of application consuming it. In the end, the last chapter discusses contributions of this thesis, outlines our next steps and gives an overview of issues and research challenges for Linked Data community.

Chapter 2

Semantic Web

The idea of the Semantic Web was created by Tim Berners-Lee in 2001. He defines the Semantic Web as "a web of data that can be processed directly and indirectly by machines" [3]. Information have well-defined meaning, better enabling computers and people to work in cooperation. Simply, Semantic Web is data about data or metadata. The Semantic Web is not a separate Web but an addition to the normal web we all know. This chapter will discuss Semantic Web and its technologies.

```
<ul>
    <li rdf:about="http://dbpedia.org/resource/London">City 1</li>
    <li rdf:about="http://dbpedia.org/resource/Prague">City 2</li>
</ul>
```

This is an example of using metada for describing semantics of hyperlink. The semantic information is provided using the *rdf:about* tag. Machines processing this code know that more information about City 1 can find at `http://dbpedia.org/resource/London`, more precisely - tag *<li>* is about resource `http://dbpedia.org/resource/London`.

2.1 Semantic Web Stack

The Semantic Web Stack illustrates the architecture of the Semantic Web and contains technologies used for creating Semantic Web applications. The stack will be discussed from bottom to top layers.

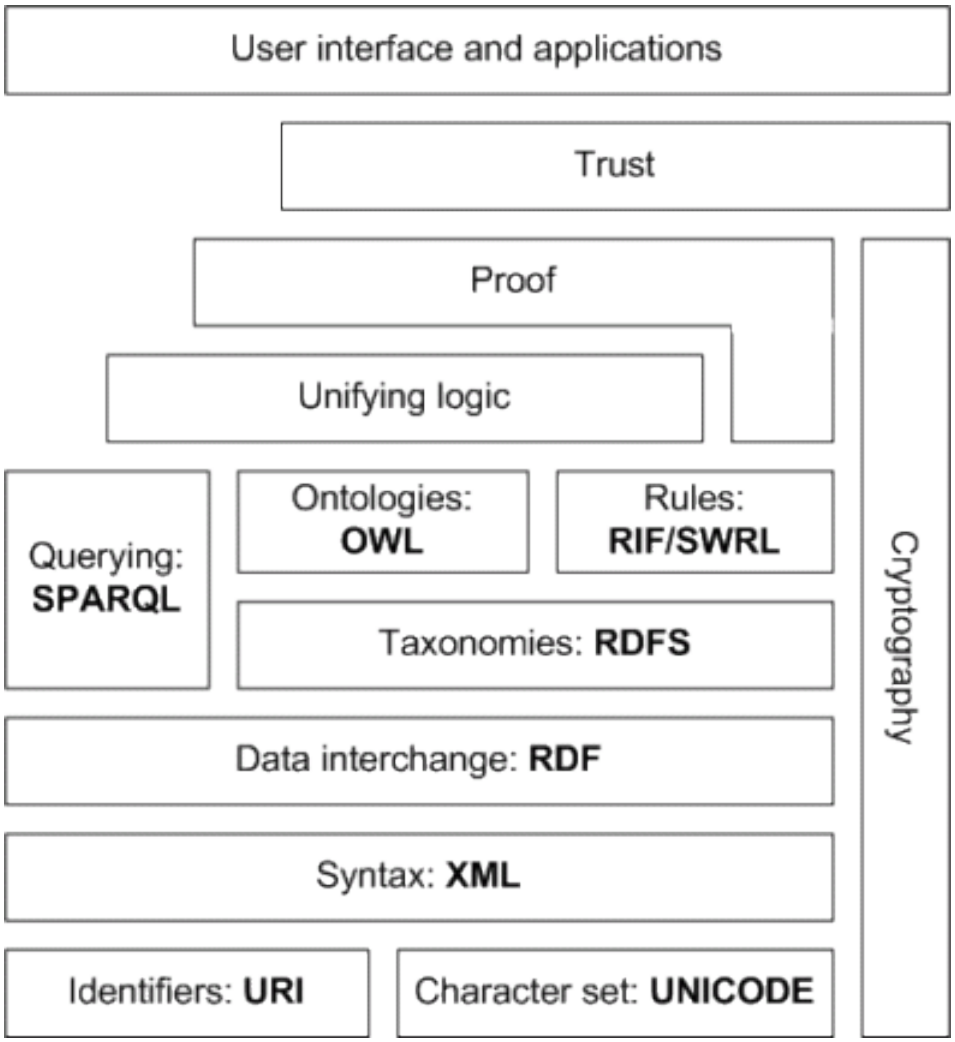


Figure 2.1: Semantic Web Stack

The bottom layers contain technologies that are well known from hypertext web and that without change provide the basis for the semantic web.

- URI, a Uniform Resource Identifier is a string of characters used to identify a name or a resource on the Internet. Semantic Web resources are uniquely identified by URI.
- Unicode is a standard for encoding most of the world existing symbols and alphabets.
- XML, Semantic Web or a 'Web of data' consist only from structured data supplemented with appropriate metadata. Therefore, XML markup language is used. The Semantic Web connects data together and XML Namespaces provide a way to use markups from more sources and to refer more sources in one document.

Technologies standardized by W3C for building semantic web applications belong to the middle layers.

- Resource Description Framework (RDF) is a framework for describing Web site's metadata. RDF use URIs for identifying resources and describe relationship between resources. Several syntactic representations are available, including XML.
- RDF Schema (RDFS) is an extension to RDF and provides a basic vocabulary for RDF. RDFS also provides type modeling laguage for describing application-specific classes and properties.
- Web Ontology Language (OWL) is a markup language for publishing and sharing data using ontologies on the World Wide Web. It extends RDFS and adds more complex constraints and more advanced constructs to describe semantics of RDF statements. Chapter 4.3 explains ontologies in more detail.
- SPARQL is a query language for searching in any RDF-based data.

Top layers contain technologies that are not yet standardized or contain just ideas that should be implemented in order to realize Semantic Web. Unifying Logic and Proof layers are undergoing active research.

2.2 Linked Data and Semantic web

The Semantic Web is a Web of Data — of dates and titles and part numbers and chemical properties and any other data one might conceive of. The collection of Semantic Web technologies (RDF, OWL, SKOS, SPARQL, etc.) provides an environment where application can query that data, draw inferences using vocabularies, etc [4]

But what is Linked Data? What is the relationship between Linked Data and the Semantic Web? The Semantic Web is the goal or end result, Linked Data provides the means to reach that goal [5]. The Semantic Web is not just about putting data on the web. It is about making links, so that a person or machine can explore the web of data [6]. And this is the task for Linked Data, Linked Data is basically about creating typed links between data from different sources. We can also say, that Linked Data is just a style of publishing data on the Web.

Linked Data will be discussed in detail in Chapter 3.

Chapter 3

Linked Data

The term Linked Data refers to a set of best practices for publishing and connecting structured data on the Web [5]. In 2006 Tim Berners-Lee outlined four principles of Linked Data in his *Design Issues: Linked Data note* [6] :

- Use URIs as names for things.
- Use HTTP URIs so that people can look up those names
- When someone look up a URI, provide useful information, using the standards (RDF, SPARQL)
- Include links to other URIs, so that they can discover more things

"*In contrast to the full-fledged Semantic Web vision, linked data is mainly about publishing structured data in RDF using URIs rather than focusing on the ontological level or inference. This simplification — just as the Web simplified the established academic approaches of hypertext systems — lowers the entry barrier for data providers, fostering widespread adoption*" [7].

This chapter will describe key Linked Data's technologies, its principles, the transformation of ordinary data into Linked Data and presents existing Linked Data sets and applications.

3.1 Linked Open Data project

Linked Open Data project [1] (LOD) started in January 2007 by Chris Bizer and Richard Cyganiak. Its goal was to identify existing datasets and convert them to RDF according to Linked Data principles and publish them on the web. Since then the amount of datasets published according to Linked Data principles started to grow fast.

[1] http://www.w3.org/wiki/SweoIG/TaskForces/CommunityProjects/LinkingOpenData

In May 2007 there were about 500 million RDF triples interlinked by over 120,000 RDF links. But in March 2008 it was already several times more. Figures below show so-called LOD cloud diagrams - images of datasets that have been published in Linked Data format by contributors to the Linking Open Data community project and other individuals and organisations.

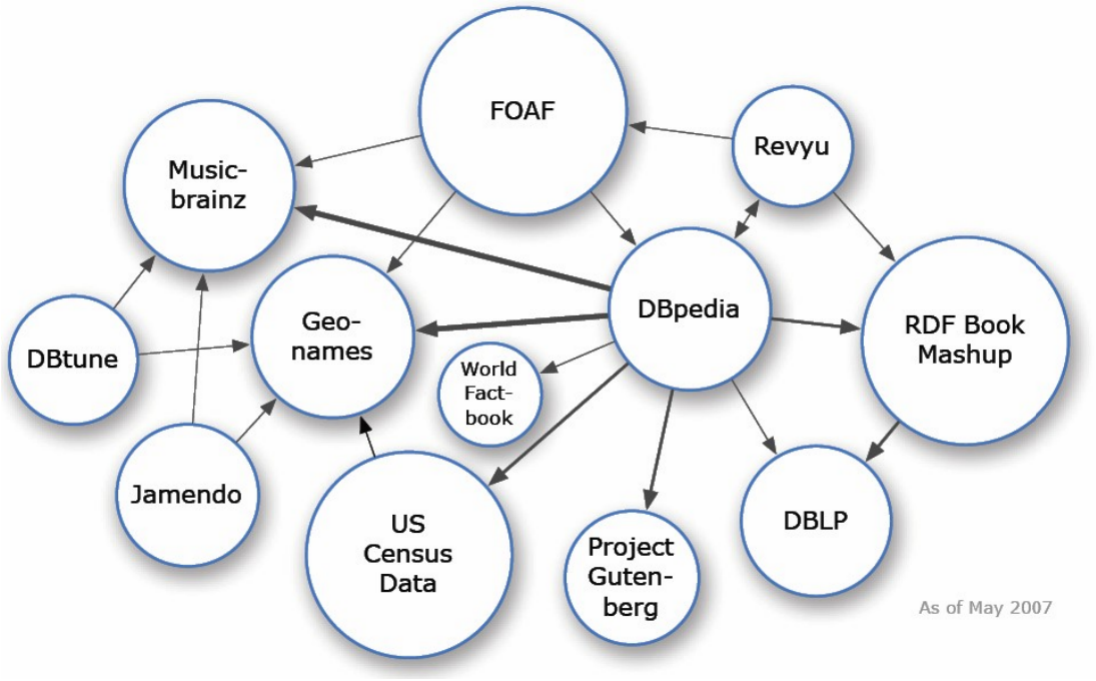


Figure 3.1: The Linking Open Data cloud diagram in May 2007. Over 500 million RDF triples. Around 120,000 RDF links between data sources. 12 Datasets.

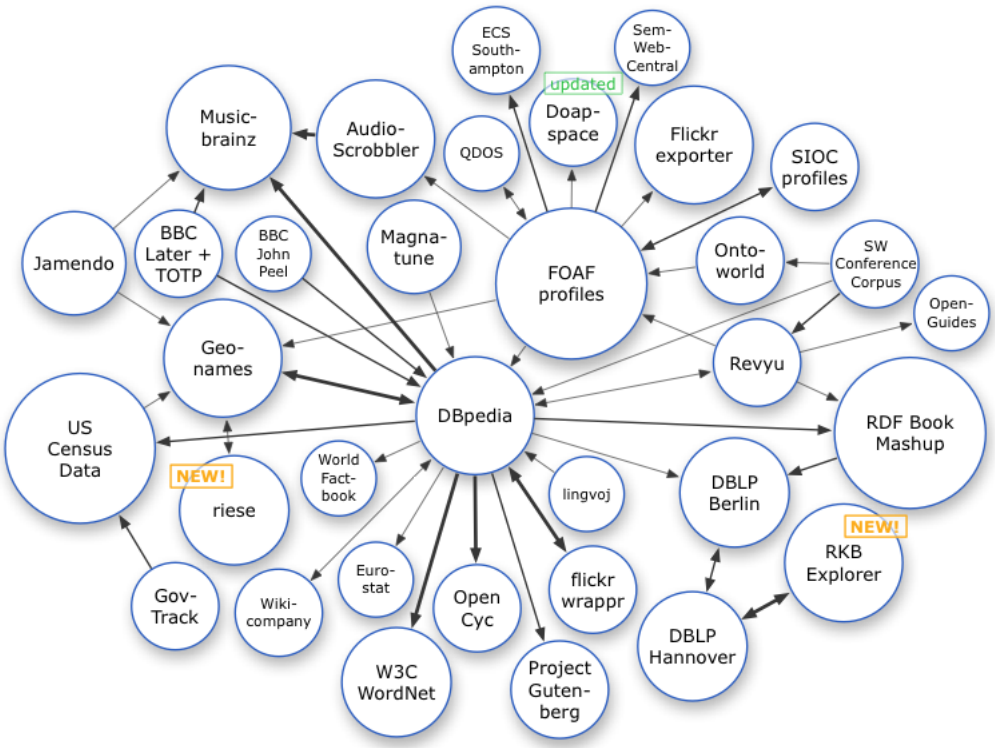


Figure 3.2: The Linking Open Data cloud diagram at March 2008. More than 2 billion RDF triples. Interlinked by around 3 million RDF links. 34 datasets.

An up to date LOD cloud diagram is available at `http://richard.cyganiak.de/2007/10/lod/`. In December 2010 there were in Linked Data cloud about 25 billion RDF triples interlinked by around 450 million RDF links in over 200 datasets [8].

Someone might point out that already a lot of public datasets accessible via Web 2.0. APIs or Web Services do exist. Nowadays these APIs are often used during Web development. The term most often referred to in this context is mashup[9]. But there is one major drawback of Web 2.0. APIs against Web of Data. It is necessity to always learn new mechanism how to access or how to work with offered data. Every Web Service is different. The main difference between the Web of Data and Web 2.0. APIs can be summarized: *"The Web of Data also opens up new possibilities for domain-specific applications. Unlike Web 2.0 mashups which work against a fixed set of data sources, Linked Data applications operate on top of an unbound, global data space. This enables them to deliver more complete answers as new data sources appear on the Web."* [5]

The Linking Open Data 2 (LOD2)[2] project co-funded by the European Commission started in September 2010 to continue the work of the Linking Open Data project. Some of the project's goals are[10] to improve coherence and quality of

[2]http://lod2.eu/

data published on the Web, close the performance gap between relational and RDF data management, establish trust on the Linked Data Web and generally lower the entrance barrier for data publishers and users.

3.2 RDF

As was mentioned earlier Resource Description Framework (RDF) is the framework for describing Web site's metadata. The basic concept of RDF is to encode (meta-) data in sets of triples, each triple being the subject, verb (or predicate) and object of an elementary sentence. Assertions are made that particular things (e.g. people, webpages, or whatever imaginable) have properties (such as 'is a sister of' or 'is created by') with certain values (another person or another webpage). Subjects and objects are each identified by a URI. The predicates are also identified by URIs, which enables anyone to define a new predicate just by defining a URI for it. Because RDF uses URIs to encode this information in a document, the URIs ensure that concepts are not just words in a document but can be tied to a unique definition that everyone can find on the Web.

The RDF data model is a syntax-neutral way of representing RDF expressions. The basic data model consists of three object types:

- **Resources** All things being described by RDF expressions are called resources. A resource may be a webpage (HTML document), a part of a webpage (a fragment) or a collection of pages, e.g. an entire website. A resource may also be an object that is not directly accessible via the web (e.g. a printed book or a person).

- **Properties** A property is a specific aspect, characteristic, attribute or relation used to describe a resource. Each property has a specific meaning and can define its permitted values, the types of resources it can describe and its relationship with other properties.

- **Statements** A statement is a specific resource together with a named property plus the value of that property.

RDF uses a particular terminology for describing various parts of statements. Specifically, the part that identifies the thing the statement is about (the webpage in this example) is called the subject. The part that identifies the property or characteristic of the subject that the statement specifies (e.g. creator, creation-date or language) is called the predicate and the part that identifies the value of that property is called the object [4]. Hence, a statement is a triple of the following form: {sub, pred, obj}

A nodes-and-arcs diagram can be used to visualize RDF statements pictorially, as shown in Figure 3.3. It represents the hierarchy of an article together with people related to the article. The subjects and objects are represented by circles, while the predicates are within the arcs.

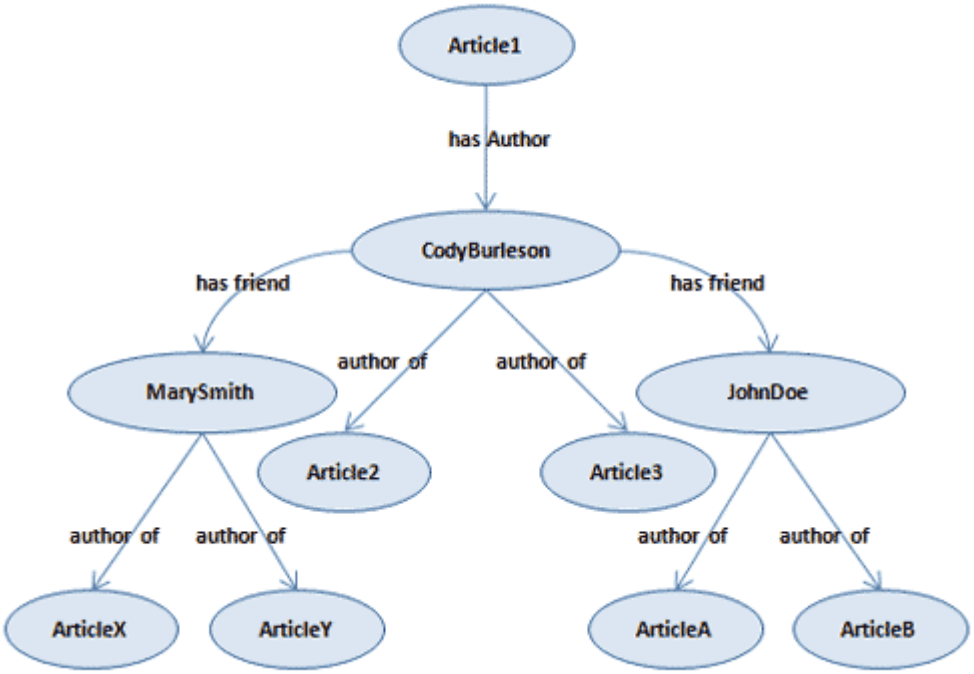


Figure 3.3: Example of RDF graph - `http://www.semanticfocus.com/media/insets/rdf-graph.png`

3.3 Ontologies

In Computer Science domain is ontology defined as: *"An ontology is a detailed model/picture/schema (can be intertwined) of a slice of reality which is based on the facts that we know about that reality. This model/picture/schema is a description of some of the things and some of the relationships between the things that are known about that reality."* [11]

Ontologies can be shared by different applications, people and databases within a domain. A domain can be an area of knowledge, like medicine or a more specific subject area like publications. Ontologies are able to specify the following kinds of concepts:

- Classes (of things)
- Relationships between classes

- Properties (attributes) of classes

There are many motivations for developing and using ontologies[12]:

- To share common understanding of the structure of information among people or software agents
- To enable reuse of domain knowledge
- To make domain assumptions explicit
- To separate domain knowledge from the operational knowledge
- To analyze domain knowledge

Within the Semantic Web it is common to, wherever possible, reuse ontologies instead of creating new ones. In order to make it as easy as possible for machines to process your data, it is best to reuse terms from well-known ontologies. Only define new terms if it is impossible to find the required terms in existing ontologies, so that redundancy is minimized.

3.4 SPARQL

The SPARQL Query Language for RDF (SPARQL) is a predominant query language for RDF graphs. SPARQL is the RDF equivalent to the relational query language SQL. Here is an example of very simple SPARQL query:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdf: <http://www.w3c.org/1999/02/22-rdf-syntax-ns#>
SELECT DISTINCT ?name ?email WHERE {
  ?person rdf:type foaf:Person.
  ?person foaf:name ?name;
          foaf:mbox ?email
}
```

The query returns the names and the email addresses of all the people in the database.

We can see similarity with SQL query syntax by having a look at SPARQL query. In the example is defined the abbreviation *foaf* (abbreviation of URI `http://xmlns.com/foaf/0.1/`) by PREFIX notation. These abbreviations or prefixes can be used later in SPARQL query to point resources without writing whole URI of resource which makes query more readable. In the example *foaf:Person* can be resolved to `http://xmlns.com/foaf/0.1/Person`.

Even though there are some similarities between SQL and SPARQL syntax, SPARQL has special constructs based on the nature of RDF data:

- **ASK.** Tests whether the RDF graph contains some data of interest.
- **DESCRIBE.** Generates an RDF description from resource(s).
- **CONSTRUCT.** Creates a custom RDF graph based on query criteria. Can be used to transform RDF data.
- **UNION.** Forms a disjunction of two RDF graph patterns. Solutions to both sides of the UNION are included in the results.

SPARQL drawbacks in comparison with SQL and XQuery as are stated in [13] :

- Lack of wide deployment. SPARQL is relatively young, and as such there are not many data stores which can be directly queried with SPARQL (as compared with SQL or XPath).
- Immaturity. As a young query language, SPARQL lacks the explicit processing model of XQuery or the decades of SQL-optimization research.
- Lack of support for transitive/hierarchical queries. SPARQL does not approach the power of, for instance, XQuery's axes.

and benefits:

- Implicit join syntax. SPARQL queries RDF graphs, which consist of various triples expressing binary relations between resources, by specifying a subgraph with certain resources replaced by variables.
- SPARQL has strong support for querying semistructured and ragged data—i.e., data with an unpredictable and unreliable structure. Variables may occur in the predicate position to query unknown relationships, and the OPTIONAL keyword provides support for querying relationships that may or may not occur in the data (a la SQL left joins).
- SPARQL is built to support queries in a networked, web environment.

3.5 Transform dataset to Linked Data

A Linked Data set is set of RDF triples. A Triple is the basic unit of RDF data. A Triple is formed of a subject, predicate and object. It is interpreted as stating

that some Subject is related to some Object by a relationship specified by the Predicate. Below is an example of RDF triple. First RDF triple 'say' that movie at the URI in the subject is same as movie at the URI in the object. Second RDF triple 'say' that director in the subject has web page at the URI in the object. There exists several serialization formats as RDF/XML, Notation3 or Turtle.

```
Subject: http://data.linkedmdb.org/page/film/122
Predicate: http://www.w3.org/2002/07/owl#sameAs
Object: http://dbpedia.org/page/Schindler%27s_List

Subject: http://data.linkedmdb.org/page/director/8477
Predicate: http://xmlns.com/foaf/0.1/page
Object: http://www.freebase.com/view/en/steven_spielberg
```

Figure 3.4: Example RDF links

In order to transform dataset to Linked Data and publish it on the web it is necessary to perform several steps or decisions:

1. How will be Linked Data used? Do you want to only publish your data as Linked Data or use them also as data source for your applications? Is it possible to transform your dataset to Linked Data only once and use it as primary data source? Or your primary data source will be still in another format (e.g. relational database) and data needed to be transformed to Linked Data continuously as your primary data source will change. Answering all previous questions in necessary to choose appropriate tools which are discussed in next sections and chapters.

2. Understanding data, find key things (people, places, etc.) and find suitable vocabularies. Well-know vocabularies (FOAF, SIOC, SKOS, vCard, Dublin Core) should be reused wherever it is possible. New terms should be defined only if they are not contained in existing vocabularies.

3. Choosing URIs for things and transformation or mapping data to Linked Data.

4. Link to other datasets (interlinking).

5. Publish data or make them accessible via SPARQL endpoint. Send pings to Sindice[3] and pingthesemanticweb.com.

All steps of dataset's transformation to Linked Data will be more discussed in following sections and chapters.

[3]http://sindice.com/

3.6 Interlinking

In order to meet one of the conditions of Linked Data principles: "Include links to other URIs, so that they can discover more things", we need to enrich our dataset with external information. This process is called Interlinking. For small datasets or RDF-files it is possible to find datasets to link against and create links manually. For larger datasets it is necessary to use some tools like Silk or LinQl. These tools will be discussed later in this section as well as all aspects of interlinking process and decisions which have to be done.

First decision which must be done is to choose against which datasets links will be created. Some popular datasets were introduced in chapter 3. A good idea is to look at the up to date LOD cloud diagram discussed in the same chapter. It is important to realize that it is not obligatory to create links only against Linked Data sets. Links can be created also against ordinary HTML web pages. For example GeoNames[4] - geographical database which covers all countries and contains over eight million placenames is very popular for creating links. This database offers rich API which delivers information (in XML of JSON format) according to place name, address, coordinates or even Wikipedia articles related to place of interest. Of course the usage of Linked Data sets is more comfortable as SPARQL endpoint can be (usually) used for asking specific information.

After choosing appropriate datasets to link against, the next step is the choice of predicates which will be used in links. Below is example of some popular generic predicates for linking:

- **owl:sameAs** - this predicate indicates that two URI references actually refer to the same thing.
- **foaf:homepage** - a homepage for some thing.
- **foaf:primaryTopic** - the primary topic of some page or document.
- **rdfs:seeAlso** - this predicate indicates a resource that might provide additional information about the subject resource.

Owl, *foaf* and *rdfs* are common abbreviations for OWL Web Ontology Language[5] , FOAF ontology[6] and RDF Schema[7] respectively. Ontologies were more discussed in the section 3.3.

The last thing which is important to point out is that links may not be created only to external datasets. It is a wise decision to enrich your own datasets with

[4]http://www.geonames.org/
[5]http://www.w3.org/TR/owl-ref/
[6]http://xmlns.com/foaf/spec/
[7]http://www.w3.org/TR/rdf-schema/

new links between 'things' inside the dataset. Data will be described better and their querying will be simpler. Interlinking tools are discussed in 4.4.

3.7 Datasets and application examples

Popular interlinking and Linked Data sets are introduced in this section together with examples of Linked Data applications.

DBpedia[8]. LOD cloud diagrams in the section 3.1 show that DBpedia is a central interlinking hub for the emerging Web of Data[14]. DBpedia knowledge base altogether consists of over 672 million pieces of information (RDF triples) out of which 286 million were extracted from the English edition of Wikipedia and 386 million were extracted from other language editions[15]. These information extracted from Wikipedia and transformed to structured form (RDF) are accessible via SPARQL endpoint. This allows to make very complicated queries like : "All soccer players, who played as goalkeeper for a club that has a stadium with more than 40.000 seats and who are born in a country with more than 10 million inhabitants". By allowing complex queries to be asked against Wikipedia content, the DBpedia knowledge base has the potential to revolutionize the access to Wikipedia[14].

GeoNames[9] is a geographical database which covers all countries and contains over eight million placenames which makes it very popular for interlinking. Each resource has its own unique URI with RDF representation. However, SPARQL is not available and database can be queried only by XML based web services.

Linked Internet Movie Database (LinkedMDB)[10] aims at publishing the first open linked data dedicated to movies, with high quality and quantity of interlinks to other LOD data sources (e.g. flickr^{TM} wrappr for movie's photos or MusicBrainz for movie's music composers) and movie-related websites.

Flickr^{TM} wrappr[11] as the name suggest the flickr wrappr uses photos posted on flickr, popular public photo repository, and creates links between them and related DBpedia records.

[8] http://www.dbpedia.org/
[9] www.geonames.org
[10] http://linkedmdb.org/
[11] http://www4.wiwiss.fu-berlin.de/flickrwrappr/

MusicBrainz[12] is an open content music database which offers public SPARQL endpoint. MusicBrainz contains information about artists, their recorded works, and the relationships between them.

DBLP Bibliography database[13] contains bibliographic information about scientific papers. The database contains more than 800.000 articles and 400.000 authors.

DrugBank[14] publishes Linked Data about over 5,000 drugs. It contains detailed information about drugs including chemical, pharmacological and pharmaceutical data; along with comprehensive drug target data such as sequence, structure, and pathway information.

More Linked Data sets can be found on:
http://www.w3.org/wiki/TaskForces/CommunityProjects/LinkingOpenData/DataSets

An example of a Linked Data application is **Revyu**[15]. Revyu is a freely accessible rating website following the Linked Data principles. The site archives ratings of every entity you can name. All these entities are represented by URIs. The reviews, written by users, are linked against DBpedia and LinkedMDB.

DBpedia Mobile[16] is a location-centric DBpedia client application for mobile devices. The application uses current GPS position of mobile device and shows nearby DBpedia resources with labels and icons. It is possible to filter displayed resources. For example only train stations on museums can be displayed. Obivously users can view information from DBpedia, but also from another interlinked datasets such as GeoNames, Revyu or Flickr.

[12] http://musicbrainz.org/
[13] http://dblp.l3s.de/d2r/, http://www4.wiwiss.fu-berlin.de/dblp/
[14] http://www4.wiwiss.fu-berlin.de/drugbank/
[15] http://revyu.com/
[16] http://wiki.dbpedia.org/DBpediaMobile

3.8 Conclusion

In this chapter the so called Linked Data principles were introduced. LOD cloud diagrams showed fast growth rate of Linked Data sets (data published according to Linked Data principles) in the past years supported by the Linked Open Data project. Real world Linked Data sets and applications consuming them were discussed as well as involved technologies like RDF, ontologies or SPARQL query language. Since a lot of developers is familiar with SQL some drawbacks and benefits of SPARQL against SQL were given.

Chapter 4

Tools

This chapter gives an overview of the tools involved in the different steps of Linked Data creation or transformation out of ordinary dataset. RDF converters which can create Linked Data from different data sources as xml, spreadsheets or txt files are discussed first. The next sections presents tools for storing RDF data (so called triple stores). Tools for mapping relational data to RDF are discussed as well because a lot of data are backed by relational databases. The section called Interlinking discuss creating links between Linked Data sets. Tools for automatic interlinking process are presented in next section. Publishing Linked Data on the web is discussed in the last section together with tools.

4.1 RDF converters

Tools and handy utilities for converting data into RDF are described in this section. These tools helps with conversion of data from application specific format into RDF representation.

NOR2O software library[1] is a library for transforming non-ontological resources to ontologies. RDF can be generated from various data sources as spreadsheets, xml or txt files.

bibtex2rdf[2] is a tool for converting bibliographic references in the BibTeX (LaTeX can generate reference lists automatically in this format) format to RDF.

jpeg2rdf[3] is an utility which extracts EXIF information from JPEG files and dumps them into into a file in an RDF/N3 representation. EXIF (Exchangeable

[1] http://mccarthy.dia.fi.upm.es/nor2o/
[2] http://www.l3s.de/ siberski/bibtex2rdf/
[3] http://simile.mit.edu/repository/RDFizers/jpeg2rdf/

image file format) is a standard that specifies the formats for images, sound, and ancillary tags used by digital cameras (including smartphones), scanners and other systems handling image and sound files recorded by digital cameras[16].

random2rdf[4] is an utility for generating random RDF graphs, because it can be hard to find real datasets that exhibit particular graph characteristics.

prefix.cc[5] is an handy utility for looking up URI prefixes. For example if we type *foaf* then URI *http://xmlns.com/foaf/0.1/* of foaf namespace is returned.

More RDF converters and utilities can be found on
http://www.w3.org/wiki/ConverterToRdf or at http://notes.3kbo.com/sparqplug.

4.2 Triple stores

A triple store is a database for the storage and retrieval RDF triples. Some triple stores solutions use as a store existing relational database engines (e.g. SQL) and some are builded as database engines from scratch.

Sesame[6] is a framework for processing RDF data. This includes parsing, storing, inferencing and querying of/over such data. It offers an API that can be connected to all leading RDF storage solutions. It can be deployed on top of a variety of storage systems (relational databases, in-memory, file systems, keyword indexers, etc.). Sesame fully supports the SPARQL query language for querying and offers access to remote RDF repositories using the exact same API as for local access.

Talis Platform[7] is a cloud-based storage for RDF data with various auxiliary services including change management, access control, index/search and SPARQL. Access is entirely over HTTP. Talis Platform is free of use for non-commercial projects.

Virtuoso Universal Server[8] is a commercial solution used for example by DBpedia. It offers SPARQL endpoint or Linked Data interface for accessing data. RDF data can be stored directly in Virtuoso or mapped on the fly from relational database.

[4] http://simile.mit.edu/repository/RDFizers/random2rdf/
[5] http://prefix.cc/
[6] http://www.openrdf.org/
[7] http://www.talis.com/platform/
[8] http://virtuoso.openlinksw.com/

Jena[9] is a Java framework for building Semantic Web applications supporting RDF API, reading and writing RDF in RDF/XML, N3 and N-Triples, OWL API, In-memory and persistent storage and SPARQL query engine.

D2R server[10] is not native RDF triple store, but database to RDF wrapper which rewrites SPARQL queries into SQL queries. D2R server is more discussed in 4.3.

More triple stores can be found on:
http://www.w3.org/2001/sw/wiki/Category:Triple_Store. Table 4.1 contains comparison of discussed triple stores. Table shows if tools are native RDF stores or if they only maps relational data to RDF. Table also shows which tools are free of use.

Tool	**Native RDF store**	**Mapping from RDB**	**Free of use**
Sesame	yes	no	yes
Talis platform	yes	no	only for non-commercial projects
Virtuoso	yes	yes	no
Jena	yes	no	yes
D2R	no	yes	yes

Table 4.1: An overview of triple stores

All previously stated triple stores are relatively new and they do not have behind them the decades of optimization research as a relational databases have. The Berlin SPARQL benchmark[11] experiment has shown that the performance of any store is not dominant for all type of queries (queries had different complexity, filter constructs, etc.). Their performance differs with the amount of stored RDF triples [17]. Some stores are faster with smaller datasets and another has better performance with larger datasets.

[9] http://jena.sourceforge.net/index.html
[10] http://www4.wiwiss.fu-berlin.de/bizer/d2r-server/
[11] http://www4.wiwiss.fu-berlin.de/bizer/BerlinSPARQLBenchmark/

4.3 Mapping relational data to RDF

A majority of dynamic Web content is backed by relational databases (RDB), and so are many enterprise systems. In 2007, it was determined that Internet accessible databases contained up to 500 times more data compared to the static web and roughly 70% of websites are backed by relational databases [18]. Linked Data is still young and an experimental technology as well as available tools. Moreover, triple stores does not achieve such performance as today's relational databases. That is why is hardly possible to begin data migration from relational databases to triple stores or to use Linked Data as primary data source. More likely scenario is that data in the relational databases as primary data source will exist together with their RDF representation used for experimental and testing purposes until technologies involved in Linked Data and Semantic web grow up. Therefore, in order to create a Linked Data dataset out of a regular relational dataset it is necessary to map it to RDF.

D2R server is a tool for publishing non-RDF relational databases on the Semantic Web[12]. D2R uses a declarative XML-based language (D2RQ Mapping Language) for mapping the content of a relational database to RDF. The mappings to RDF can be generated automatically. Database tables are mapped on RDF classes and columns on RDF properties using 'fictional' ontology generated by D2R. The mapping rules can be customized by user and fictional ontology replaced by real ones. The relationships between database tables (e.g. foreign keys) is also possible to map to valid RDF, because mapping language supports SQL queries in mapping process. This means, that the values from the tables to which foreign keys point are used instead of the values of foreign keys. The data insertion is supported by extension of D2R, but it is in an experimental phase.

D2R Server works with Oracle, MySQL, PostgreSQL, Microsoft SQL Server, and any SQL-92 compatible database. Microsoft Access can be used with some restrictions. Mapped data can be accessed via SPARQL. Our experience with mapping a relational dataset to RDF is more described in 5.4.

Triplify is a simplistic but effective approach to publish Linked Data from relational databases [19]. Triplify is based on mapping HTTP-URI requests onto relational database queries and can be used with many widely installed Web applications. Triplify transforms the resulting relations into RDF triples and also publishes this data on the Web as Linked Data. However, this approach does not support SPARQL querying.

[12] http://www4.wiwiss.fu-berlin.de/bizer/d2r-server/

Virtuoso same like D2R server is a RDF data publishing tool which also offers mapping relational data to RDF. Virtuoso uses a declarative Meta Schema Language for defining the mapping of SQL data to RDF ontologies.

RDBToOnto is a tool for designing and implementation of ontologies from relational databases[13]. Ontologies are created by taking advantage of both the database schema and the data, and more specifically through identification of taxonomies hidden in the data[20]. RDBToOnto has been evaluated on a set of 50 databases from different domains.

Overview of mapping approaches and another mapping tools can be found in [21].

4.4 Interlinking tools

Interlinkings tools differs in a many ways. Some are domain independent some can be applied only for a certain domains of the real world (e.g. publications, music datasets)[22]. For users is important degree of automation. Some tools are completely automatic and work as black box, other tools need manual work. Important is also the data access (SQL or SPARQL queries, local copies of datasets) same as format of input (RDF, XML, SQL database) or output data.

Silk is a link discovery framework for the Web of Data[14]. Silk works over SPARQL endpoints and is domain independent. Declarative language (Silk Link Specification Language - Silk LSL) based on XML is used for specifying which types of RDF links should be discovered between data sources as well as which conditions data instances must fulfill in order to be interlinked[23]. Different transformation functions can be used in order to prepare items before they will be compared. Silk supports a many string comparison techniques, numerical and date similarity measures to compare dates or geographical measure for computing distance between two points. Result of the comparison is the score which can be weighted and combined by aggregation functions. The output RDF triples may contain any RDF predicate specified in Silk-LSL configuration file, not only common *owl:sameAs*.

The framework also contains a tool (Silk Workbench) for evaluating the generated RDF triples in order to improve interlinking process and a graphical editor

[13] http://www.tao-project.eu/researchanddevelopment/demosanddownloads/RDBToOnto.html
[14] http://www4.wiwiss.fu-berlin.de/bizer/silk/

which enables the user to easily create and edit link specifications. Documentation is prepared very-well and contains real world examples.

Our experiences from using Silk in a real word project are discussed in section 5.6.

LinQuer (Linkage Query Writer)[15] is a tool for generating SQL queries for semantic link discovery over relational data and currently supports MySQL and IBM DB2. The LinQuer framework consists of LinQL, a language for specification of linkage requirements; a web interface and an API for translating LinQL queries to standard SQL queries; an interface that assists users in writing LinQL queries[24]. A link specification defines the conditions that two given values must satisfy before a link can be established between them. To create such links, the framework provides several different methods like ones based on synonyms, hyponyms, and a variety of string matching methods. The problem with this is that external datasets are way less available in a relational format than RDF dumps or SPARQL endpoints. The tool was used to interlink the Linked Internet Movie Database to DBpedia.

RDF-AI[16] is a domain independent tool for merging RDF datasets or for creating links between them according to user configuration. RDF-AI contains modules for preprocessing (translation via Google Translation API[17] can be performed), matching (string comparison or word relation can be specified), fusion, interlinking and post processing.

GNAT[18] and **CaMiCatzee**[19] are examples of domain specific tools. First tool is used for creating links between music datasets, while second is used for interlinking multimedia data. An overview of interlinking tools can be found in [22]. The table 4.2 shows the summary of tool's features.

[15] http://dblab.cs.toronto.edu/project/linquer/
[16] http://code.google.com/p/rdfai/
[17] http://code.google.com/p/google-api-translate-java/
[18] http://sourceforge.net/projects/motools/
[19] http://sw.joanneum.at/CaMiCatzee/

Tool	Domain	Automation	Data access	Output
Silk	any	semi	SPARQL	file with RDF triples
LinQuer	any	semi	SQL queries	SQL entries
RDF-AI	any	semi	Jena framework	merged dataset or link list
GNAT	music datasets	automatic	music files	RDF statements (owl:sameAs)
CaMiCatzee	multimedia data	automatic	FOAF access	rdfs:seeAlso in XHTML page

Table 4.2: An overview of interlinking tools' features

4.5 Maintaining Links

Datasets in LOD cloud are subject to change. Links created during interlinking process can become invalid when resources are moved, removed or updated. However, there is no widely accepted standard or best practise how to deal with broken links and current approach is to ignore this problem or leave it for applications to handle it. Interlinked resources are the major benefits of Linked Data. So there is a opinion that LOD data sources should provide the highest possible degree of link integrity in order to relieve applications from this issue, similar do databases that provide mechanisms to preserve referential integrity in their data [25].

The solutions which are not based on any central repository, such as PingtheSemanticWeb[20] or Sindice[21] will be discussed.

DSNotify is one of the proposed solutions[25] to handle this issue. DSNotify[22] is a generic change detection framework for Linked Data sources that informs data-consuming actors about the various types of events (create, remove, move, update) that can occur in data sources. It has two basic scenarios of usage. First is that two or more remote Linked Data sources using DSNotify add-on are subscribed to each other. Modifications of their local datasets are then exchanged between them. The second usage is only at a local Linked Data source. DSNotify access periodically remote Linked Data sources interlinked with local data source and uses indexing infrastructure to track the resource changes.

[20] http://pingthesemanticweb.com/
[21] http://sindice.com/
[22] http://dsnotify.org/

Web of Data - Link Maintenance Protocol (WOD-LMP) is a solution proposed by authors of *Silk - A Link Discovery Framework for the Web of Data* [23]. Authors propose a protocol for synchronizing and maintaining links between Linked Data sources. The implementation of the WOD-LMP[23] protocol is based on SOAP. The protocol supports exchange of changed, deleted or newly added links between subscribers. For example link source sends list of RDF links to the target source so that target source can track incoming links and consider to set back-links. Providing list of the changes occurred within a specified time period is also supported. WOD-LMP protocol is used for maintaining links between DBpedia and DrugBank.

4.6 Publishing Linked Data on the Web

The tools either serve the content of RDF stores as Linked Data on the Web (e.g. Virtuoso) or provide Linked Data views over non-RDF legacy data sources (e.g. D2R)[5]. Within Linked Data applications the accepted standard is to publish the data both as HTML (in order to create a view for users with a HTML-browser) and as RDF (to access the data and in order to browse the data with a Semantic Web browser). Below are examples of common URI's patterns for serving HTML and RDF views of data:

```
Thing: http://dbpedia.org/resource/Prague
RDF data: http://dbpedia.org/data/Prague
HTML page: http://dbpedia.org/page/Prague

Thing: http://revyu.com/people/teddypolar/about/
RDF data: http://revyu.com/people/teddypolar/about/rdf
HTML page: http://revyu.com/people/teddypolar/about/html
```

Figure 4.1: Common URI's patters

The first URL of both examples is an address of a resource. The second URL is an address of RDF data and third is address of HTML view. Publishing tools shield publishers from dealing with the technical details such as content negotiation. So if semantic browsers (which accept RDF headers) access the first URL they will be automatically send to the second URL, ordinary web browser to the third URL.

Various tools can be used for publishing Linked Data. Some of previously discussed triple stores are also able to publish Linked Data (D2R, Virtuoso).

[23] http://www4.wiwiss.fu-berlin.de/bizer/silk/wodlmp/

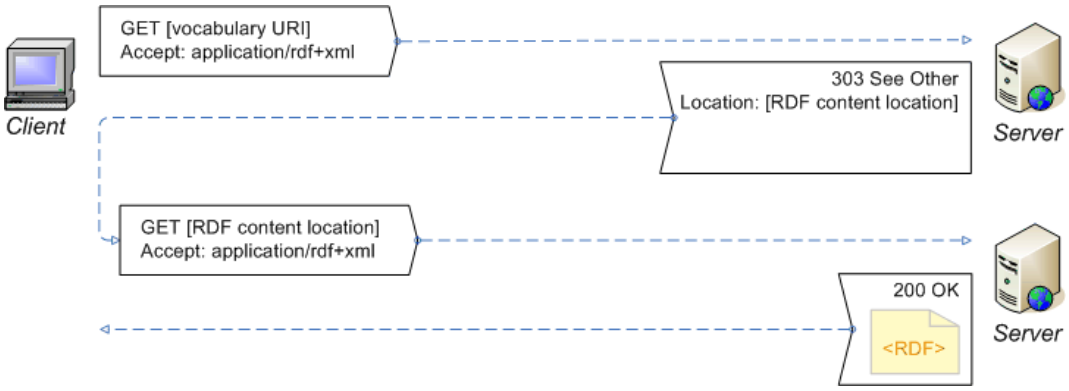


Figure 4.2: Content negotiation. Image obtained from http://www4.wiwiss.fu-berlin.de/bizer/pub/LinkedDataTutorial/, tutorial for publishing Linked Data on the Web.

Another tools like **Paget Framework** [24] or **Pubby**[25] use another triple stores as a data source. The Paget Framework serves up RDF from static files or from the Talis Platform. Pubby is a Java web application which turns a SPARQL endpoint into a Linked Data server by rewriting URI requests into SPARQL DESCRIBE queries against the underlying RDF store. Besides RDF, Pubby also provides a simple HTML view over the data store and takes care of handling 303 redirects and content negotiation between the two representations. Table 4.3 shows brief overview of discussed publishing tools.

Tool	**Data source**	**Open source**
D2R	Oracle, MySQL, PostgreSQL, Microsoft SQL Server, any SQL-92 compatible database, Microsoft Access	yes
Virtuoso	Oracle, MySQL, Microsoft SQL server, PostgreeSQL, Informix, Sybase, ODBC and JDBC accessible data sources	no/commercial
Paget Framework	static file, Talis Platform	yes
Pubby	SPARQL endpoint	yes

Table 4.3: An overview of mapping tools

4.7 Conclusion

This chapter gave an overview of the tools for different steps of Linked Data creation. We discussed tools for creation Linked Data out of spreadsheets or txt

[24] http://code.google.com/p/paget/
[25] http://www4.wiwiss.fu-berlin.de/pubby/

files (RDF converters), tools for Linked Data storage (triple stores) or tools for publishing Linked Data on the Web.

We introduced tools for mapping relational data into RDF because a majority of dynamic Web content is backed by relational databases. There are exist already enough tools for mapping relational databases to RDF. The problem is that every tool has different approach for mapping and developer needs to learn it again from the beginning.

The value of Web of Data heavily depends on the quality and amount of links between data sources. We introduced tools for automatic interlinking process for increasing amount of links. We discussed possible solutions for keeping integrity of links because resources on the Web of Data can change in time and links can become invalid. However, proposed solutions are not very widespread.

Chapter 5

Creating Linked Data

In this chapter, we identify a suitable data source and describe its transformation to Linked Data. We also describe its enrichment with an external Linked Data source. Finally, we discuss publishing the created Linked Data on the Internet.

5.1 Identification of the data sources

We have searched for open government information which can be visualized in an interesting way. Because Czech Republic has serious problems with corruption and overpricing of public contracts we have focused on data about public contracts. All contracts over a certain price must be published by law. Problem is that a lot of government institutions submit their contracts on their own web sites and in different data formats. Fortunately, there exists central point - ISVZ database[1]. This database contains information about all public contract submitted by government institutions since 2006. The database contains information about approximately 50,000 public contracts, submitted by over 5,000 government institutions supplied by over 12,000 private companies and grows every day. Each record contains submitter and supplier of a public contract, price, date, number of bids, etc.

With data from the ISVZ database, an interesting tool could be created. It could put submitting public contracts under public supervision. However, these data are presented only in a very simple form. Basically there is only a list of public contracts sorted by date or name of supplier or submitter. Searching can be done by name of a submitter or supplier, their organization numbers, date or contract name. More complicated and also interested searching have to be done manually. Sometimes, it is almost impossible, for example : searching of contracts with value over 1 billion CZK which had only one bid, searching of

[1] http://www.isvzus.cz

contracts from some geographical area, searching for most significant suppliers or submitters of public contracts , etc. This way of searching is more interesting for public supervision, because it allows to find suspicious or non-standard public contracts. Furthermore, potential users are mostly more interested in contracts in place they live.

Therefore, we decided to develop an application which allows advanced searching together with visualization results on the map. Moreover, map will allow to show connections between suppliers and submitters. Submitters which emit a lot of public contracts should have suppliers from all country. But if they have only suppliers from same town or they have still same suppliers, there could be a reasonable suspicion about lobby to officials who decide about the winner of public contracts or who determine the winning conditions.

Before we can start to develop our map application we must get also data from ARES database[2] served by Interior Ministry. This database contains public information about economic subjects in Czech Republic. We need these data because it contains exact addresses and additional information like stock holders, corporate directors, etc. With these information we can find conflicts of interest. For example, a person which decides about public contracts at some government institution is also a member of a private company. And this private company is bidding for public contracts from that government institution. However, we are limited with storing and publishing information about members of suppliers or submitters. The law prohibits to store and to publish information about persons which can lead to their identification without their permission. Obviously we do not have and we can not have their permission. So we can not download, store and publish information like names, birth numbers or addresses of supplier's or submitter's members. But it does not change the fact that we are still able to reveal conflicts of interest. We must only visit ARES database to know the real name of person we are interested in.

Finally we need to use Google Geocoding API[3] to obtain coordinates from addresses we have from ARES database so we can show subjects on the map.

5.2 Retrieving data

Before starting work on the map application we created a simple web crawler downloading data from previously mentioned databases. Obtaining data from ARES database is easy and straightforward because it offers services returning information about suppliers or submitters in XML by their organizational num-

[2] http://wwwinfo.mfcr.cz/ares/

[3] http://code.google.com/apis/maps/documentation/geocoding/

ber or company name. Google Geocoding API also offers XML or JSON output format returning coordinates and more additional information about a place specified by address. Unfortunately this is not case of ISVZ database. Web interface is focused only on human processing, content is not delivered in any structured data format. This brings us some complications:

- **Incomplete data.** Presented data itself are incorrect, incomplete and contain mistakes. A lot of contracts are missing some records such as price, proposed price, supplier or submitter organizational number or values of prices of contracts are zero. For human users, it is easy to recognize that these records are missing, but not true for machine reading. A Web crawler can not simply detect if records are really missing, empty, incorrect or they are placed on different place in the document than they are supposed to be. If there are some wrong or incorrect information after crawling (e.g. zero price) it is hard to recognize where the problem is. Presented data could have been really incorrect. But also presented data could have been formatted or presented in different way than web crawler expects.

- **Different formatting of presented data.** Data are presented in HTML tables. This is an overview representation for human users, but totally inappropriate for machine reading. A contract notice is a complicated process. Different kinds of public contracts requires different documents, administration and procedures. There is also a lot of exceptions. Therefore, presented data about public contracts are different and have different formatting. Some information about one public contract does not have to be presented about another public contract. Information is also usually presented in the format *property: value*. Problem is that in some cases is property in one column and value in second column. But sometimes property with value are only in one column of HTML table.

- **Wrong HTML formatting.** HTML document itself is sometimes not well-formatted which can bring errors to a crawling process.

The aim of this thesis is not to create sophisticated web crawler with wide options of configuration. That is why we decided to create simple, light-weight web crawler focused on scraping data form ISVZ database and parsing XML obtained from ARES database and Google Geocoding API. During development we had to take into account the fact, that the amount of requests against ARES database and Google Geocoding API is limited. ARES database allow only 1,000 requests during the day and 5,000 requests at night. Google Geocoding API offers 2,500

requests per day. ISVZ database has no limits. Therefore, we decided to separate scraping ISVZ database from using ARES database and Google Geocoding API.

Scraping ISVZ database is divided into these steps:

- The last day in which public contracts were scraped from ISVZ is saved in our database. The crawler takes that date when scraping ISVZ starts and downloads web page which contains a list of public contracts for next day. The crawler is using a web page which allows to search public contracts by date.
- The crawler goes through a list of contracts and stores links for web pages with individual contracts for that day.
- The crawler opens a web page of every contract and scrapes information. A submitter or supplier of contract is created in database, if they do not already exist.
- Every public contract has one or more subcontracts. During scraping information about contract, crawler is also searching for links to pages containing information about subcontracts. If they are found, crawler opens them and scraps information about subcontracts as well.
- Information about public contracts are saved to our database and 'last scraping date' is updated when all links for web pages with public contracts information for one day are processed. If 'last scraping date' is less than actual day, crawler continues from the beginning.

Downloading data from ARES database and Google Geocoding API is divided into these steps:

- The crawler takes a first supplier or submitter which does not have downloaded data from ARES and Google Geocoding API.
- If a submitter or supplier does not have an organizational number, crawler uses ARES service which allows to obtain it by a supplier or submitter name. If obtaining of organizational number was successful crawler continues with next step. If not, crawler continues with penultimate step.
- Data about supplier or submitter are downloaded from ARES by their organizational number. First crawler tries to download extended information. These information contains information about members of supplier or submitter, shareholders or if company is owned by secret owner. However,

these information are not available for every supplier or submitter. If they are not available crawler tries to download basic information. These information contains only basics about supplier or submitter, like name or address. These information are already available from ISVZ database. But we consider information from ARES reliable.

- The crawler tries to download coordinates according to supplier's or submitters's address from Google Geocoding API.
- The crawler saves information to our database and continues from beginning, if suppliers or submitters which does not have downloaded data from ARES and Google Geocoding API are still in database.

The web crawler is implemented in .NET (C#) and using HTML Agility Pack library[4] for scraping data from web page. The library is HTML parser that builds a read/write DOM and supports XPATH querying. The parser is very tolerant with "real world" malformed HTML. The object model is very similar to what proposes System.Xml, but for HTML documents (or streams). With this library is easier to find requested data despite of they are often in different places in html table. The web crawler also uses dotConnect for MySQL library[5]. This library allows to use LINQ technology for querying and updating mySQL database.

As was said, we decided for simple crawler. So there is no options so set. User can only press button for scraping ISVZ database or obtaining information from ARES database and Google Geocoding API. Information about processing are shown same as errors which can appear (e.g. when Google Geocoding API does not find coordinates for provided address or price of contract is not found).

Sometimes happened that some information was not available for the crawler. In this case we went through the web page and made appropriate changes in the scraping process. With this approach we were able to adapt and improve scraping process on different structure of web pages presenting data about public contracts. The current situation is that all errors which occurs during crawling process are caused by missing information in the web page. Not because crawler can not find it. Over 99% of ISVZ web pages were crawled with no missing information. Another question is quality of crawled data. Here are some statistics:

- 2% to 3% of public contracts have defined none or zero price of contract.
- 25% of subcontracts have defined none or zero proposed price by submitter.

[4] http://htmlagilitypack.codeplex.com/
[5] http://www.devart.com/dotconnect/

- 1.5% of subcontracts have defined none or zero number of bids.
- 2% of submitters and 30% of suppliers do not have known their organizational number (after attempt to obtain it from ARES database by their name).
- Obtaining coordinates by address was not successful at 4% of suppliers and at 5% of submitters.

As we can see except organizational numbers of companies and proposed prices of subcontracts are crawled data in a relatively good quality.

5.3 Related decisions

Before crawling started, we needed to decide if we transform data from crawling process directly to RDF or store them in a relational database. MySQL database was chosen for several reasons. First is that we want to compare performance and overall comfort and simplicity of developing application with mySQL database as data store with RDF data stores. Creation of RDF from relational database is easy with tools like D2R server. Second reason is that we crawl a lot of information for which does not exist appropriate vocabularies. We could create our own vocabulary with everything we need, but we believe that more elegant way is to crawl as much information as we can and simply change only mapping of relational data to RDF according to progress of development vocabularies we need. Moreover, if performance of SPARQL endpoint of RDF triple store would not be sufficient for applications consuming our data, we can still use traditional mySQL database.

Figure 5.1 shows database schema:

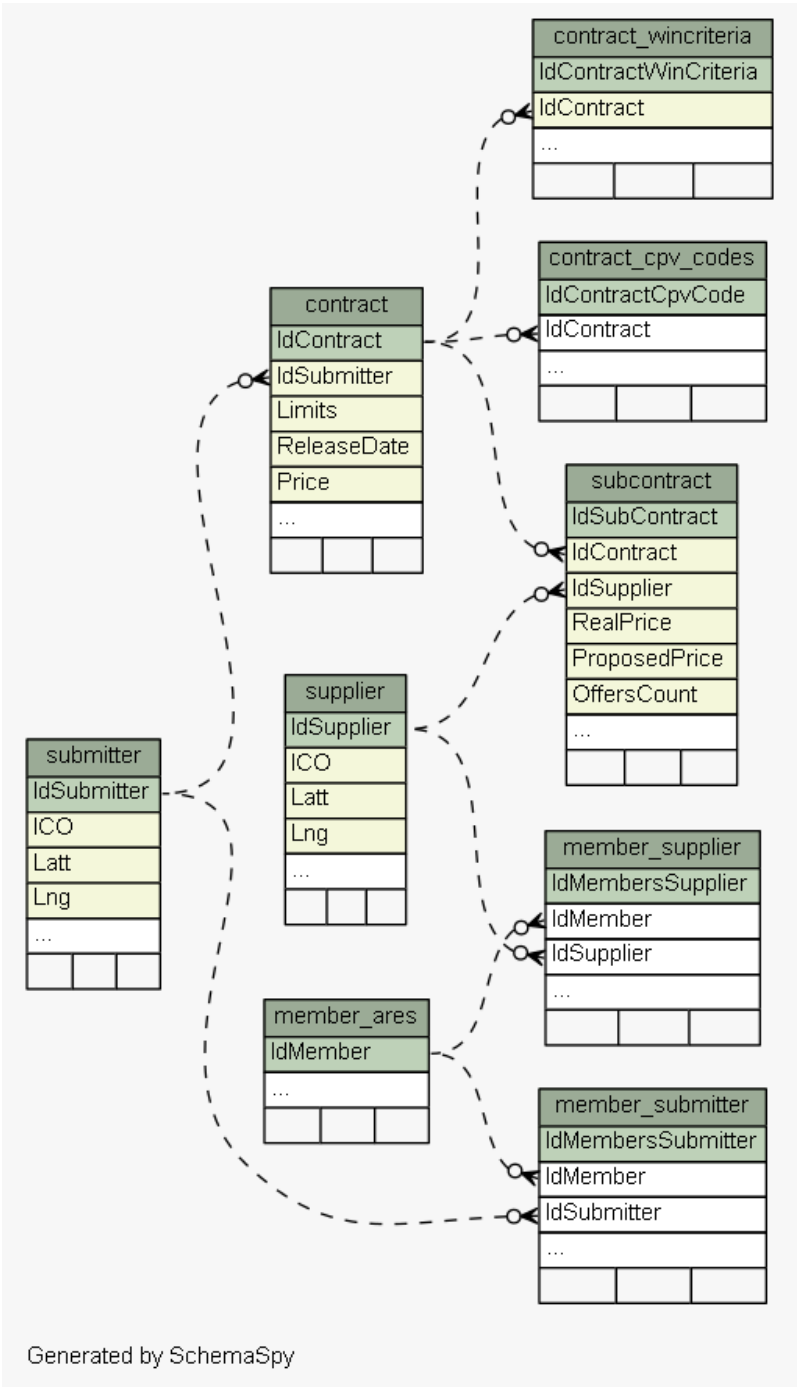


Figure 5.1: mySQL database schema

Schema is quite straightforward. Submitters submit contracts. Each contract has one or more subcontracts supplied by suppliers. Each contract has also criteria important for winning the contract. Table *member_ares* contains persons which are members of suppliers or submitters. One person can be member of more suppliers or submitters. Tables *member_submitter* and *member_supplier* assigns members to submitters and suppliers and contains information about position of members in company or institution. Table *contract_cpv_codes* contains

so called CPV codes related to contracts. CPV is an acronym for Common Procurement Vocabulary. This vocabulary has been developed by the European Union for public procurement. CPV codes are a standardised vocabulary to describe procurement notices to help procurement-responsibles to classify procurements consistently, and to help service and product suppliers find procurements of interest[16]. Below is an example of a CPV code:

```
71356200 Technical assistance services
```

5.4 Mapping the dataset to RDF

We used D2R server to map our relational data to RDF. Because D2R server does a lot of work for us with auto-generate mapping feature, we need to adapt auto generated mapping file. D2R server creates RDF classes and properties according the names of database tables and their columns, e.g. property *contract_url* for the column with name *url* in the table contract. We had to change it to classes and properties from ontologies we want to use (FOAF Ontology, Public Contract Ontology etc.). Addition of conditions to the mapping file was also necessary as we do not want to map table columns with empty values. If table contained foreign key we used D2R's join construct. This construct causes that values from tables to which foreign key points ares used instead of values of foreign key. D2R server also allows us to set our own URI patterns to 'things' (and specify these 'things' ourselves). For example this could be URI pattern for submitter of public contract : *http://www.myserver.com/submitter/orgNumber* .

We used following vocabularies for map dataset to RDF :

- **GoodRelations Ontology**[6] is a standardized vocabulary (also known as "schema", "data dictionary", or "ontology") for description of products and their features and prices, stores and opening hours, payment options and other company data. We used *BusinessEntity* class for suppliers and submitters of public contracts and class *PriceSpecification* for specification of public contracts' prices.
- **FOAF Ontology**[7] is an ontology describing persons, their activities, social networks and relations to other people and objects. We needed this ontology for mapping members of suppliers and submitters of public contracts.

[6] http://purl.org/goodrelations/v1#
[7] http://xmlns.com/foaf/0.1/

- **Public Contracts Ontology**[8] is an ontology for describing contracts in public procurement.
- **Public Contracts Ontology - Czech Module**[9] is an extension of Public Contracts Ontology given by the specifics of the Czech Republic.
- **VCard Ontology**[10] models and represents vCards in RDF. VCard is a file format standard for electronic business cards. vCards are often attached to e-mail messages, but can be exchanged in other ways, such as on the World Wide Web or Instant Messaging. They can contain name and address information, phone numbers, e-mail addresses, URLs, logos, photographs, and even audio clips[16]. This ontology is used for specification of addresses.
- **Basic Geo Vocabulary**[11] is a vocabulary for representing latitude, longitude, and altitude information. Application showing public contracts on map will be created - this vocabulary is used for mapping latitude and longitude.

Several properties and classes missed during the creation of mapping relational database to RDF. We have decided to extend Public Contract Ontology with missing properties and classes for better description of public procurement. The Contract class was extended by following properties:

- **description**: the description of the public contract.
- **subContractsCount**: number of subcontracts of the contract.
- **propsedPrice**: proposed price of the contract or subcontract by the submitter.
- **contact, email, phoneNumber, fax**: contact information. We did not use Person class from FOAF ontology, because these contacts do not have to belong to same person.
- **winCriteria**: criteria for winning the contract. There are possible two values: *EconomicallyMostAdvantageous* and *LowestPrice*. Economically the most advantageous bid according to decision criteria defined by next property win the contract in the first case. The bid with the lowest price win the contract in case of *LowestPrice* value.
- **decisionCriteria**: criteria for deciding about contract winner. This property points to instance of **DecisionCriteria** class. This class has only two properties: **criteria** and **weight**. These properties defines criterium with its importance (in percents) for winning the contract.

[8] http://purl.org/procurement#
[9] http://purl.org/procurement/czech#
[10] http://www.w3.org/2001/vcard-rdf/3.0#
[11] http://www.w3.org/2003/01/geo/wgs84_pos#

We created also class **SubContract**, because a lot of public contracts are divided into sub contracts. This class inherits from class Contract and contains only **parentContract** property. The property points to the parent contract of the sub contract. There was not necessary to add another properties, because sub contracts have same characteristics as contracts.

Next we created the class **BusinessMember** describing members of a company. The class contains following properties:

- **person**: a reference to instance of Person class from FOAF ontology containing more information about the business member.
- **businessEntity**: a reference to the instace of BusinessEntity from GoodRelations Ontology containing more information about company to which business member belongs.
- **position**: a position of the business member in the company.
- **inPositionFrom/To, memberShipFromFrom/To**: the time during which a person is in position and in the company.

Finally we extended Person class from FOAF Ontology by **personNumber** property and BusinessEntity class from GoodRelations Ontology by **hasSecretShares** property. First property gives to person unique number. Second property indicates if company has secret shares (secret owners).

After we extended Public Contracts Ontology we could have finished mapping relational database to RDF. Below are several examples of D2R mapping:

```
map:Contracts a d2rq:ClassMap;
    d2rq:dataStorage map:database;
    d2rq:uriPattern "contracts/@@contract.IdISVZUS@@";
    d2rq:class pc:Contract;
    .
```

This is an example of the mapping relational database table on Contract class from public contracts ontology (pc prefix). Each public contract has its own unique URI defined by URI pattern construct: *d2rq:uriPattern*.

```
map:contract_Homepage a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map:Contracts;
    d2rq:property pc:url;
    d2rq:uriColumn "contract.URL";
```

This example shows the mapping of the relational database column onto RDF property. Column *URL* is mapped onto property *pc:url*. There must be also specified into which class mapping this property mapping belongs (RDF property belongs to RDF class) . This is done by *d2rq:belongsToClassMap* construct.

```
map:contract_Subcontracts a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map:Contracts;
    d2rq:property pc:subContractsCount;
    d2rq:sqlExpression "Select Count(*) From subcontract
                        where IdContract = contract.IdContract";
    d2rq:datatype xsd:int;
.

map:contract_email a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map:Contracts;
    d2rq:property pc:email;
    d2rq:condition "contract.Email != ''";
    d2rq:column "contract.Email";
    d2rq:datatype xsd:string;
.

map:contract_Submitter a d2rq:PropertyBridge;
    d2rq:belongsToClassMap map:Contracts;
    d2rq:property pc:submitter;
    d2rq:refersToClassMap map:Submitters;
    d2rq:join "contract.IdSubmitter => submitter.IdSubmitter";
    .
```

These examples show more complicated mapping of relational table columns. First one uses SQL expression for counting subcontracts of contract because we do not have this information stored in database column. Second one shows usage of conditions. Because we do not want to map columns which contains empty or NULL values we used a condition that prevents mapping empty Email columns. Third one shows how to 'connect' RDF properties with classes. Contract class from public contracts ontology contains a property *submitter* which refers to *BusinessEntity* class (GoodRelations Ontology) representing contracts submitters. *D2rq:refersToClassMap* construct refers to a class mapping which maps a relational database table representing submitters onto *BusinessEntity* class. There must be also specified how will be the 'connection' between *submitter* property and instance of *BusinessEntity* class established. This is done by *d2rq:join* construct. This construct says that table contract contains foreign key IdSubmitter which refers to table submitter.

5.5 SPARQL endpoint

D2R server provides also SPARQL endpoint for querying data. But during testing it we had performance problems with more complicated queries. First SPARQL query (shown below) works correctly and results are shown after several seconds. It shows 10,000 contracts' URLs out of approximately 50,000.

```
PREFIX pc: <http://purl.org/procurement#>
SELECT ?subj WHERE {
    ?subj ?x  pc:Contract.
} LIMIT 10000
```

But performance problems come when the queries start to be more complex. Next query shows only 10 contracts' URLs and their names.

```
PREFIX pc: <http://purl.org/procurement#>
SELECT * WHERE {
    ?subj ?x  pc:Contract.
    ?subj rdfs:label ?label.
} LIMIT 10
```

After executing this query we must wait considerably longer time to see the results (tens of seconds). Linked Data application discussed in chapter 6 will need significantly more complicated queries with a lot of filters. Furthermore we want to provide public SPARQL endpoint for querying our data so we have decided to try Sesame triple store[12]. We must point out that the problem is not caused by mySQL database at background. Direct querying of mySQL database has no performance problems.

Sesame offers several types of repositories for storing RDF triples. We tested in memory repository (persistent and non persistent) and mySQL repository. As was mentioned in the section 4.2 Sesame is native triple store and does not map relational data as D2R server does. So first we need to obtain data to import into Sesame. D2R server helps with this as D2R can dump data mapped from relational database into file in chosen format (N3, N-TRIPLE etc.). These dumped data can be imported to Sesame.

Sesame offers web interface for managing triple store repositories so import data is easy and straightforward. Unfortunately Sesame does not show progress of data import. This is not a problem when using small datasets or when we use memory as a store. But it becomes to be a problem in case of large dataset and mySQL as repository. Import of our dataset which consists over 2 million

[12]http://www.openrdf.org/

RDF triples takes only tens of seconds in case of memory repository, but over 10 hours in case of mySQL repository. During import there is no response about progress. Only working hard drive and growing tables in mySQL database show that something is happening. Sesame does not store data in mySQL database in naive way, but create a lot of tables with a lot of indexes. This is perhaps the reason why insertion takes so much time. Process of data insertion can not be affected in any way by the user. It works as black box. Structure of the database tables is not anywhere documented and is very unclear. So there is no other possibility to insert data to the database more effectively.

A memory repository is an ideal solution if there is enough free memory available. Even very complicated SPARQL queries are very fast. In our case (over 2 million RDF triples) Sesame needs about 600MB RAM. Memory repository can be persistent, so if we restart server data will not be lost and performance is almost same. Sesame only starts slower as it needs to load into memory our dataset from hard drive.

Performance is poor even though import of data to MySQL repository takes so much time and a lot of tables and indexes are created. Even first query we showed before which takes D2R server only several seconds takes Sesame about 20 seconds. But second query which takes D2R server between one and two minutes takes Sesame again about 20 seconds. Sesame process better more complicated queries, but simple queries takes too much time. Anyway it is not possible to use Sesame SPARQL endpoint in our application. Waiting times are too long and queries we will need to process are much more complicated. We tested available tools from the view of ordinary developer which may not have access to powerful server and which usually develops applications on local machine. Machine with only average speed was used for our testing. But processing of so simple queries which takes so much time shows that a lot of work on optimalization need to be done. More precise benchmark can be found in chapter 7.

Sesame has one major drawback in comparison with D2R server. Data in Sesame are not up to date till we import them again, because we have data in relational database and new data are added also to this database. Fortunately Sesame imports only RDF triples which are not already in its database, so we can easily export all data from D2R server and import them to Sesame. But this can be a solution only if we use memory as a store or have smaller amount of RDF triples. Using this approach with mySQL database as a store is very time consuming with larger amount of RDF triples. In this case is necessary some mechanism for exporting only newly added data from relational database.

Next thing we point out is that there could be slight differences in SPARQL implementations. Following SPARQL query works with D2R, but not with Sesame. More precisely, query works, no syntax error occurs, but result is empty.

```
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX pc: <http://purl.org/procurement#>
SELECT * WHERE {
  ?subj ?x pc:submitter.
  ?subj rdfs:label ?label.
  FILTER( regex( ?label, 'Praha' , 'i' ) ).
}
```

In sesame results are showed only if inside FILTER construct is applied *str* function:

```
FILTER( regex( str(?label), 'Praha' , 'i' ) ).
```

Last thing we need to discuss is diacritics. Czech language have some special characters, for example: ž,č,ř,á,í. These special characters sometimes causing problems to various applications. D2R server provides SPARQL endpoint for 'machine-querying' (e.g. by PHP scripts) and AJAX-based SPARQL Explorer for human users. SPARQL endpoint does not have problems with diacritics, but SPARQL explorer for human users does. Query which contains some character with diacritics does not work. No error occurs, but letters get malformed after query submit and no results are returned. Sesame does not work with letters with diacritics at all. SPARQL endpoint for machine querying and GUI based SPARQL endpoint for human users does not work. No results are always returned.

5.6 Interlinking

In this section will be described interlinking our dataset with external data source and difficulties we had during this process. Because our dataset contains tens of thousands of public contracts, suppliers, submitters and another additional information it is obvious that interlinking process can not be done manually.

We have chosen Silk from tools discussed in section 4.4 to help us with interlinking process. Silk was chosen for several reasons - Silk is domain independent, for data access uses SPARQL endpoint which we need to access DBpedia and has very well-prepared documentation with a real world examples. As was explained in section 4.4, with *Silk Link Specification Language* (Silk-LSL) is possible to specify which types of RDF links should be discovered between data sources as

well as conditions which data items must fulfill in order to be interlinked. We have set for us two goals:

First goal was to interlink submitters representing cities (municipal offices) with additional information about cities they represent from DBpedia. More specifically, information about : population, Wikipedia web page, city web page, leader party and leader name (if these information will be available). First we needed to do is separating submitters representing cities from another submitters. Names of these submitters (*rdfs:label* property) contain patterns like 'Municipal office Brno', 'City Ostrava' or only 'Municipal office'. With Silk-LSL is possible to restrict results obtained from SPARQL endpoints using SPARQL clauses in Silk-LSL. So we were able to easily filter submitters for using in interlinking process. After we had filtered submitters we could have used names of cities from their addresses for comparison against records from DBpedia. Small problem was that some submitters have in their addresses only part of the city. For example, there were Prague 1 instead of Prague. For this Silk-LSL offers many transformation functions as *removeBlanks*, *replace*, *lowerCase*, *upperCase* or regular expressions which can transform input. With them we were able to extract name of city from the address or from the name of submitter.

In previous chapter were discussed D2R server's problems with performance of SPARQL endpoint and problems with diacritics which Sesame have. First we have used Sesame SPARQL endpoint because of better performance. But a lot of Czech cities have in their name some letters with diacritics. This caused that Sesame has returned empty results on Silk's SPARQL queries and a lot of submitters were missing. Then we have tried also D2R. Fortunately D2R has no performance problems with queries on submitters. So all submitters could have been used in our interlinking process.

Unfortunately Silk has one serious limitation which prevented us from full achieving our first goal. Below is an example how is possible to restrict results obtained from SPARQL endpoints using SPARQL clauses in Silk-LSL.

```
<SourceDataset dataSource="sesame" var="a">
    <RestrictTo>
        ?a rdf:type gr:BusinessEntity
    </RestrictTo>
</SourceDataset>
<TargetDataset dataSource="dbpedia" var="b">
    <RestrictTo>
        ?b dbpedia-owl:country dbpedia:Czech_Republic
    </RestrictTo>
```

```
</TargetDataset>
```

First restriction (*gr:BusinessEntity* class) filters only submitters from our dataset. For simplicity is not shown the restriction on submitters representing cities. Second restriction is for DBpedia. We had to use *dbpedia:Czech_Republic* class for filtering because incomprehensibly a lot of Czech cities does not belong to *dbpedia-owl:Cities_and_towns_in_the_Czech_Republic* or even *dbpedia-owl:City* DBpedia category.

Next example shows definition of comparison of RDF subjects obtained from SPARQL endpoints:

```
<Compare metric="levenshtein">
    <TransformInput function="lowerCase">
        <Input path="?a/vcard:Locality"/>
    </TransformInput>
    <TransformInput function="lowerCase">
        <Input path="?b/rdfs:label"/>
    </TransformInput>
</Compare>
```

Submitter's city name is compared with value of label property of record from DBpedia (lower case function is performed on values before comparison). If values are same, then link is crated. Type of the link can be specified like this:

```
<LinkType>owl:sameAs</LinkType>
```

And here is the problem. Only subjects filtered by restriction rules can go to output. It is possible to specify what properties (*rdfs:label* and *vcard:Locality* in our case) will be used in comparison rules. But there is no possibility to specify (for example) *foaf:page* property to be used for output. So now we are able to do RDF triples like this:

```
<http://localhost:2020/resource/submitter/7>
<http://www.w3.org/2002/07/owl#sameAs>
<http://dbpedia.org/resource/Liberec>
```

This kind of RDF triples are correct and we can use them in order to enrich our dataset, but it is not exactly what we wanted. We would need to specify property which will be used for output, for example *foaf:page*. Then we would be able to take an object from this RDF triple:

```
<http://dbpedia.org/resource/Liberec>
<http://xmlns.com/foaf/0.1/page>
<http://www.liberec.cz/>
```

and use it with submitter. Result would look like this:

```
<http://localhost:2020/resource/submitter/7>
<http://xmlns.com/foaf/0.1/page>
<http://www.liberec.cz/>
```

But as was said this is not possible. At least we were able to create *owl:sameAs* links between submitters and DBpedia records about them.

Second goal was to use Silk to enrich our dataset with links between items from our dataset. Names of submitters or suppliers are often misspelled or their names are bit different. For example *Company Ltd.*, *Company Ltd* or *Company, Ltd.* The web crawler can identify some basic typos during scraping process, but can not identify more complicated ones. String comparison measures which Silk-LSL offers could be used for solving these problems with typos. We found settings which worked best for us after some labor and experimentation. But we were only partially successful, because the resulting output file contained a lot of false RDF triples. We must say it is not fault of Silk. These are typical false triples:

- Some larger cities have several parts, for example Prague 1, Prague 2, etc. and each part has own municipal office. Names differ only by number. This is typo for every string comparison measure.
- A lot of villages have very similar names which again considered as typos.
- Companies uses a lot acronyms or abbreviations for name and they are often very similar. It is again considered like typos by Silk.

Silk works very well for the problem mentioned earlier, *Company Ltd.* x *Company, Ltd.*, different count of spaces between words and other similar cases. What we needed to do was to separate correct RDF triples from wrong. Silk Workbench would be an ideal solution. It is possible to mark RDF triples as correct or incorrect after they are generated. But there is no way how to export them! In Silk Workbench is possible to see process of evaluation of triples. Feature of labeling triples is intended only for saving history of evaluation. This helps with improving settings file for interlinking, because it is possible to see if evaluation of triples is getting better or worse. So user can easily see if last changes was correct or wrong. Please see figure below.

Figure 5.2: Silk Workbench tool - example of RDF triples evaluation.

First expanded RDF triple on the figure is clearly correct - company name differs only because of one more space. Second expanded triple is wrong, it belongs to different town parts of Prague. On the right side are buttons which can mark if triples are correct or incorrect. But as we said there is no way how to export triples marked as correct. Only solution is to go through output file, open URL of every triple and check if they really belong to same supplier or submitter. But this takes so much time that the results it brings does not worth the time spent over it. It must be said that second goal is not met.

Even though we were not successful with our goals, we still consider Silk as best tool for interlinking, because Silk-LSL is very flexible, domain independent and Silk Workbench is great tool for increasing quality of a link specification. Silk is also part of Linking Open Data 2[13] project, which proves its quality.

We have files with RDF triples containing links between our dataset and DBpedia. Here comes to surface problem of D2R server. There is no way how to merge triples from interlinking process with our dataset. D2R has prototype extension for SPARQL/Update[14] which allows to update database with SPARQL queries. Problem is that it was tested only with one type of experimental mapping and it is intended only for experimental purposes. But from our point of view this type of extension is not necessary. Ideal solution would be if SPARQL endpoint during querying took into account also RDF triples from specified external files. However, this is not possible now. We can only offer them for download. Adding RDF triples from interlinking process to Sesame is not a problem.

[13] http://lod2.eu/
[14] http://d2rqupdate.cs.technion.ac.il/

5.7 Publish the RDF dataset

Let us recapitalize the Linked Data principles:

- Use URIs as names for things.
- Use HTTP URIs so that people can look up those names
- When someone look up a URI, provide useful information, using the standards (RDF, SPARQL)
- Include links to other URIs, so that they can discover more things

We see that the last step we need to perform is publishing our data. We have already found ontologies fitting our data, mapped them to RDF and finally we created links to external data sources.

D2R server used for mapping relational database to RDF is also capable to present both HTML and RDF views upon mapped data. D2R server follows common URI's patterns for serving RDF and HTML views, for example:

`http://gd.projekty.ms.mff.cuni.cz:2020/resource/contracts/50000176`
`http://gd.projekty.ms.mff.cuni.cz:2020/data/contracts/50000176`
`http://gd.projekty.ms.mff.cuni.cz:2020/page/contracts/50000176`

First URL is address of resource, in this case it is public contract. Second URL is address of RDF data for Semantic web browsers and third URL is HTML view for ordinary web browsers.

5.8 Conclusion

In this chapter was described the whole process of creating Linked Data, what difficulties it can bring and what decisions which must be done during this process. We found suitable data sources, created web crawler and download data into mySQL database. We found ontologies fitting our dataset and mapped data to RDF. We enriched data with external data sources and published our data with D2R server and Sesame. Our dataset contains approximately 2.5 millon RDF triples and still grows as every day are submitted new public contracts. Each resource has its own unique URL and can be queried by SPARQL. Now we are ready to create an application consuming Linked Data.

Chapter 6

Linked Data application

In this chapter is described the development of the application upon data discussed in previous chapter. The application uses Google Maps JavaScript API V3[1] for visualization data on map. Data will be delivered to the application in JSON format by PHP. We will experiment with several data sources: mySQL relation database, D2R server mapping data from mySQL and Sesame triple store. Data from D2R server and Sesame triple store will be accessed via their SPARQL endpoints.

6.1 Application features

The aim of our application is to visualize data acquired from ISVZ and ARES databases (and in the future from more government databases). Users will have wide possibilities of searching, filtering and querying these data. As was discussed in section 5.1 current ways of work with data are very limited. The application will be also enriched on the fly by data from DBpedia via SPARQL endpoint.

As was mentioned earlier, Czech Republic has serious problems with corruption in public procurement. More options how to work with data acquired from public procurement we provide, more public supervision will be pointed at public procurement. Public contracts will became more transparent and less expensive. Moreover it creates pressure on government institutions to provide more data and in more friendly format for machine processing. For example now are provided only winners of public contracts, other bids with reasons why they were not chosen are not public (this knowledge would be very valuable because very often wins the bid which does not have the lowest price). Providing data in machine readable form also supports further development of similar kind of applications.

Major features of the application:

[1] http://code.google.com/apis/maps/documentation/javascript/

- Showing contracts, submitters and suppliers on the map by their addresses.
- Showing connections between contracts, submitters and suppliers. After user clicks on icon of some subject it will show connections (lines on map) with his suppliers/submitters/contracts. For example, if user clicks on the submitter icon on the map all his suppliers will appear connected with lines. This allows to reveal if some submitters submit contracts to still same suppliers.
- Details about subject will be showed after clicking on it - details of contract, supplier or submitter.
- Extensive filter options. Contracts will be possible to filter by their price, number of bids, supplier or submitter of contract, date, difference between proposed price by submitter and real win price, name of contract and another options given by specifics of public procurement in Czech Republic. Suppliers and submitter will be possible to filter by their name, organizational number, town, and number or value of submitted or acquired contracts. Option for searching only in actual map bounds.
- Map will be independent on data source. Data will be delivered by PHP scripts in JSON format same for different data sources (mySQL, SPARQL endpoint).

Here is the list of the major application use cases:

- Searching for the most expensive public contracts.
- Searching for the biggest submitters or suppliers by amount of contracts or by overall value of contracts
- Searching contracts having only one bid.
- Showing and searching contracts in user's neighborhood. Users are often interested how are money for public contracts spent in place they live.
- Exploring another contracts of suppliers or submitters which are suspected by manipulation, corruption or another illegal behavior in public procurement or exploring connections between them.
- Showing contracts exceeding proposed price by their submitter.
- Revealing suppliers cartel - contracts from some submitters are won by still same suppliers. Suppliers should be also from all country not only from submitters' neighborhood.

- Showing suppliers with secret shares.
- Revealing big difference of price in same kind of public contracts e.g. supplies of computers, medical supplies etc.

We believe our application can contribute to eliminate problems in public procurement in Czech Republic and has potential to reach a big number of users. Major groups of users are:

- Journalists involved in revealing corruption or any another suspicious circumstances in public procurement.
- People interested in spending of public money by town municipal office or any another government institution (elementary or high schools, universities etc.). Map will help them to focus only on submitters from place they live as this is usually most important for them.
- People dissatisfied with situation of public procurement in Czech Republic.
- Businessmen or company owners who want to participate in public procurement. With our tool they can easily search for contracts they are interested in.
- Businessmen or company owners unsuccessful in public procurement having doubts about tender of public contract they do not win. With our tool they can easy reveal that for example submitter of contract they did not win is giving contracts to still same suppliers. Our tool can help them with decision to bring legal action or to file an appeal on results of public procurement.

6.2 Application architecture

The schema 6.1 shows architecture of the application. The schema visualizes the whole process of collecting data, their transformation and consumption by our application.

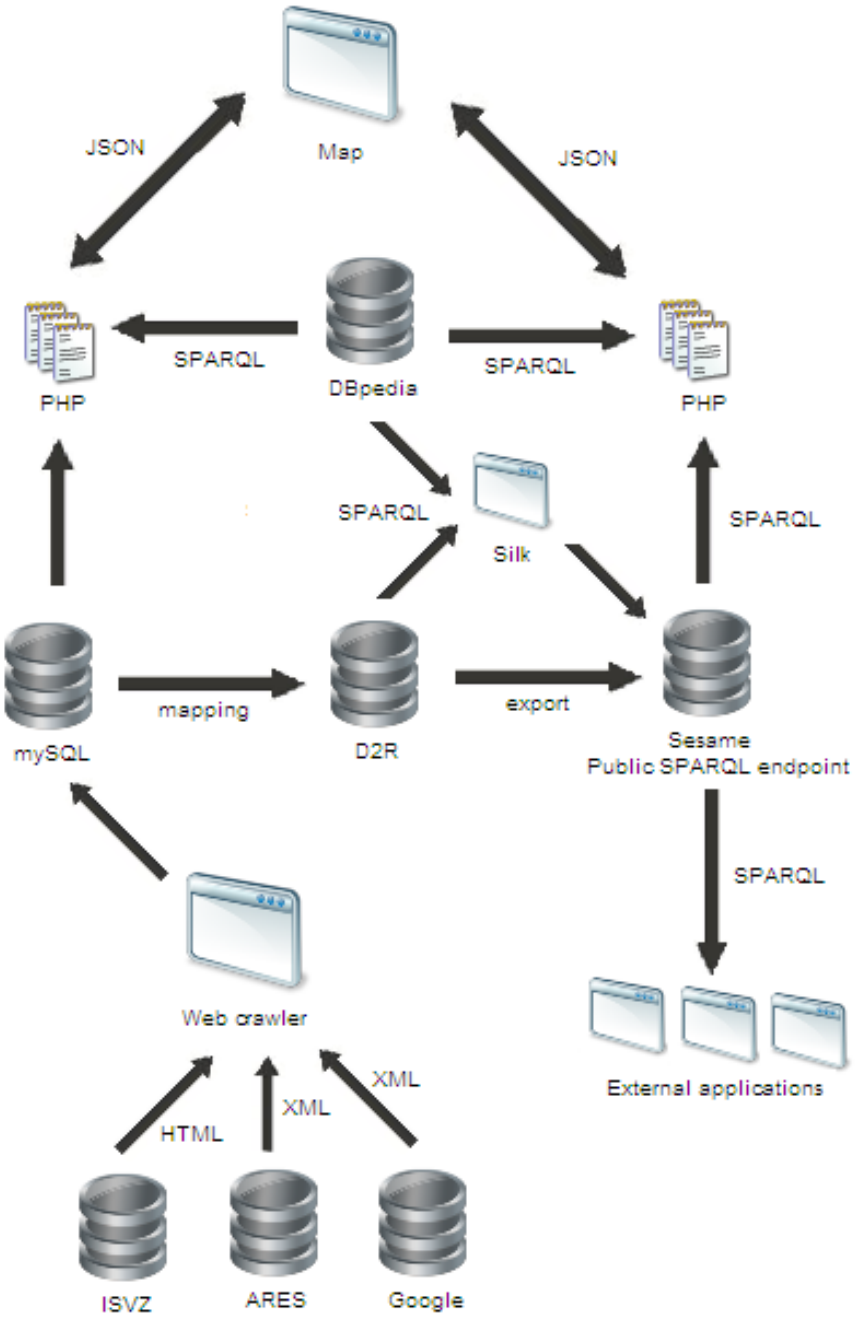


Figure 6.1: application architecture

The web crawler scraps data from HTML pages from ISVZ web and also uses ARES and Google Geocoding XML web services. These data are then stored in

mySQL database. Data are mapped by D2R server into RDF. Because we had performance problems with D2R's SPARQL endpoint we exported data from D2R server and import them to Sesame. Sesame provides public SPARQL endpoint which can be used by external applications. Silk created links between our dataset and DBpedia. Silk used D2R and DBpedia SPARQL endpoints. Sesame SPARQL endpoint was not used for interliking because of Sesame has problems with diacritics. Created links were then imported into Sesame.

Application uses AJAX technology for obtaining data from PHP scripts. Data are delivered in JSON format. PHP scripts obtain data directly from mySQL database or from Sesame's SPARQL endpoint and encodes them to JSON data format. These data are then enriched by information from DBpedia and returned to application.

6.3 Data retrieval

The application retrieves data from PHP scripts in JSON format which is same for different data sources. First we made PHP scripts which obtain data in standard way from mySQL database, transform them to JSON and return to application. Then we created PHP scripts which obtain data from SPARQL endpoint. For querying it we used simple PHP library[2] which allows us to query SPARQL endpoints in same manner as usual mySQL database. So only difference between querying mySQL database and SPARQL endpoints is only query itself. Processing parameters passed to PHP script and processing retrieved data and transforming them to JSON are similar. Obviously creation of mySQL queries was faster and simpler, because of more previous experiences. But we must say that after some time became querying by SPARQL intuitive and even faster and simpler than by SQL.

First we tested our application against D2R's SPARQL endpoint. The application worked correctly, but as was mentioned in section 5.5 D2R server has performance problems with querying our data set. The application was responding slowly and overall user experience was not good. Main benefit of using D2R server is that data are always up to date, because D2R only maps relational database which stores data from the web crawler. Main drawback is that we can not add simply new RDF triples we have from interlinking process, because D2R server does not support insertion or merging with new RDF data. If D2R could merge mapped data with data from interlinking process and had better performance it would be ideal solution for us.

[2] http://graphite.ecs.soton.ac.uk/sparqllib

The application worked with no problems against Sesame SPARQL endpoint. We were also able to add easily new RDF triples we had from interlinking process. One drawback is that data in Sesame are not up to date until new import is performed. Second drawback is that Sesame has problems with diacritics. If searching containing letters with diacritics is performed, then empty result is returned.

6.4 Enrichment with data from DBpedia

One of our goals is to enrich application with data from external linked data source in on-the-fly manner. Unfortunately there is not enough suitable data sources. Showing pictures from the flickr wrapper according to place where is located public contract, supplier or submitter is not interested feature for users of our application. Only DBpedia offers suitable information. If submitter of public contract is clicked on the map and submitter is city which has record on DBpedia then additional information are downloaded via DBpedia's SPARQL endpoint and showed in application. These additional information contains population, mayor or leader of the city and his party, home page or Wikipedia page.

Let's consider we are searching for Prague on DBpedia[3]. First issue we found is that DBpedia does not officially support Czech language so names of cities (predicate rdfs:label) are not in Czech. In case of Prague we are lucky, because in Norwegian or Finnish is name of Prague same like in Czech (Praha). Fortunately other cities have their names usually same like in Czech in most of supported languages.

The most simple SPARQL query would like like this: (we will not show prefixes, as they are automatically defined in DBpedia SPARQL query explorer)

```
SELECT DISTINCT * WHERE {
   ?subj rdfs:label ?label.
    FILTER(str(?label) = 'Praha') .
} LIMIT 1
```

This query exceeds limit of maximum execution time because of amount of data on DBpedia and performance of SPARQL. We must first reduce the number of predicates in which we are searching for Prague. We tried to filter first by category *dbpedia-owl:Cities_and_towns_in_the_Czech_Republic* and it worked perfect for Prague. But as was mention in section 5.6 incomprehensibly a lot of Czech cities does not belong to this category. Same situation is with *dbpedia-owl:City* category. So we used as filter only *dbpedia:Czech_Republic*. This cate-

[3]DBPedia SPARQL query explorer: http://dbpedia.org/snorql/

gory does not reduce number of predicates so much as previous mentioned filter, but it is sufficient for execute the query. Our final query looks like this:

```
SELECT DISTINCT ?subj, ?totalPop , ?leaderName , ?leaderParty ,
                ?homepage, ?wikiPage, ?mayor
    WHERE {
      ?subj dbpedia-owl:country dbpedia:Czech_Republic.
      ?subj rdfs:label ?label.
      FILTER(str(?label) = 'Praha') .

      OPTIONAL{ ?subj dbpedia-owl:populationTotal ?totalPop }
      OPTIONAL{ ?subj dbpprop:leaderName ?leaderName }
      OPTIONAL{ ?subj dbpprop:leaderParty ?leaderParty }
      OPTIONAL{ ?subj dbpprop:mayor ?mayor }
      OPTIONAL{ ?subj foaf:homepage ?homepage }
      OPTIONAL{ ?subj foaf:page ?wikiPage }

    } LIMIT 1
```

We must have used OPTIONAL constructs, because information we look for are not available for every city. If we did not use OPTIONAL construct there would be an empty result if city miss one of the RDF triple we look for.

Another issue which had to be solved was how to show name or title of information (total population, leader name ...) we obtain from DBpedia. Obviously we can not show name of RDF properties (dbpprop:leaderName, dbpedia-owl:populationTotal etc.). But labels of these properties are defined only in English. Currently there is no general solution how to solve this. We had to make manual translation (basically just a lot of conditions in source code) for each property which is showed in our application.

We believe that spirit of Linked Data is to easily enrich application with additional data. But it is not sustainable to always manually translate labels of RDF properties. In our situation we can not show all properties connected to city we are searching for - it is hundreds of items and we can not translate them all manually. Let's consider following RDF triple :

```
<http://gd.projekty.ms.mff.cuni.cz:2020/data/submitter/461>
<http://purl.org/procurement#url>
<http://wwwinfo.mfcr.cz/cgi-bin/ares/darv_or.cgi?ico=25622684>
```

And here is definition of *url* property from Public Procurememnt Ontology:

```
pc:url a rdf:Property ;
  rdfs:label "Odkaz"@cs, "Hyperlink"@en ;
  rdfs:comment "Odkaz na další informace o veřejné zakázce"@cs,
    "A hyperlink to additional information about a public contract"@en ;
  rdfs:subPropertyOf rdfs:seeAlso ;
  rdfs:domain pc:Contract ;
  rdfs:range rdf:Resource ;
  rdfs:isDefinedBy pc: .
```

There is only Czech and English label and description. Applications in other languages can not use these provided labels and manual translation as we also did is necessary. Obviously creators of ontologies can not provide labels of RDF classes and properties in every language. Therefore, there should exist some mechanism how to extend ontologies with labels and comments in another languages. And what is more important - to make it easily reusable.

Last issue we must point out is that DBpedia has also problems with diacritics. DBpedia returns an empty result set if searching for city which contains in name some letters with diacritics is performed.

6.5 Conclusion

Our goal was to create application consuming data from common data source (mySQL database) and from SPARQL endpoint and compare both approaches from the view of developer. Once we already have access to Linked Data is application development same as development against traditional data source from the view of complexity and time. To get familiar and intuitive with SPARQL is very fast and programming queries is from our point of view even easier than SQL queries. Moreover, there is one major benefit of using SPARQL. All proprietary APIs are always different, same as relational databases. They have different structure, tables, foreign keys etc. For using them is necessary to learn new APIs or structure of database again and again from scratch. This is not a case of SPARQL. Once we get familiar with some ontology (FOAF, Public contracts ontology, GoodRelations Ontology , ...) we can query every endpoint using that ontology no matter what data it represents.

Created application is also enriched on the fly with data from DBpedia. During this enrichment with data from DBpedia we found an issue which should be solved. Labels of RDF properties are usually only in language of creator of ontology and in English. So manual translation of labels was necessary after obtaining data from DBpedia's SPARQL endpoint. We believe that there should be some

mechanism how extend existing ontologies by labels in other languages and what is important to make it public and easily reusable.

Here are some examples of the searchings which we can easily done with our application (answers reflect situation on July 2011) :

- How many public contracts with price over 2.5 billion CZK (about 100 million euro) with only one bid exists? Answer: 46!.
- How much public contracts submitted Prague (municipal office)? And how much of them had only one bid? Prague submitted 865 public contract. 508 of them had only one bid.
- How much suppliers with secret shares won some public contract (these companies has secret owners, they can be owned by politicians who decide about public contracts)? Answer: 174. 6 of them won contracts with value over 500 million CZK (about 20 million euro).

It would not be possible to find answers on a previously stated examples or it would be extremely laborious without our application.

Chapter 7

Application benchmark

In the previous chapter was described how we created the application consuming data from common data source (mySQL database) and from D2R's and Sesame's SPARQL endpoints. Their performance was briefly discussed in sections 5.5 and 6.3, especially the problems with SPARQL endpoints performance. This chapter will describe the more precise benchmark of PHP scripts which delivers data to our application. We will simulate the behavior of a common user and test server performance during workload with different amount of users. Apache JMeter[1] was chosen as a testing tool. Apache JMeter is a Java desktop application which can be used to simulate a heavy load on a server, network or object to test its strength or to analyze overall performance under different load types.

7.1 Benchmark definition

We will do only benchmark of PHP scripts delivering data to our map application. The map is a common web page, but its logic contains javascript code which is not executed by Apache JMeter. Moreover, simulation of filling search fields in a web page is very problematic. That is why we test only PHP scripts. The PHP scripts will be called in the way how they are called by the map with appropriate parameters. Therefore, benchmark will reflect the real usage of map. We will test both PHP scripts which deliver data from mySQL database and from SPARQL endpoints of D2R server and Sesame. Sesame will be tested when uses memory as a repository.

The experiment was conducted on server with configuration: processor: Intel Xeon E5640 2.67GHz; memory: 4GB RAM, SCSI HDD, MS Windows Server 2008 R2 Standard (64bit). We used the following releases of the systems under test:

[1] http://jakarta.apache.org/

1. **MySQL database** version 5.5.15.
2. **Sesame** version 2.4. (memory repository) with Tomcat version 7 as HTTP interface.
3. **D2R server** version 0.7.

The configuration of PC performing benchmark is following: processor: Intel Mobile Core 2 Duo T7200 2.00GHz; memory: 4GB RAM, MS Windows 7 64-bit SP1. PC is connected to Internet with speeds 25Mbps for download and 1.5Mbps for upload.

Before we can start with benchmark we must define behavior of a common user. Bellow is the list of actions which can be considered as behavior of a common user.

1. Search 1: Show contracts submitted this year (2011) on the map. The user watches map from 10 to 20 seconds and does another search (next step).
2. Search 2: Show contracts with value over 100 million CZK (about 4 millon Euro).
3. Detail 1: The user clicks 5 times on contracts in 30 seconds (details about contracts are showed after click). Then user reads the details of the last clicked contract from 20 to 30 seconds.
4. Detail 2: The user watches detail of supplier of contract from previous step from 10 to 20 seconds.
5. Search 3: Show contracts from Prague, with price over 100 millon CZK (about 4 millon Euro) and with only one bidder.
6. Detail 3: The user clicks 5 times on contracts in 20 seconds. Then the user reads details of last clicked contract from 20 to 30 seconds.
7. Search 4: Show submitters from some town.
8. Detail 4: The user clicks 5 times on submitters in 30 seconds. Then user reads details of last clicked submitter from 20 to 30 seconds.
9. Detail 5: Showing detail of supplier from one of contracts submitted by the submitter from the previous step and reading supplier's details from 20 to 30 seconds.
10. Search 5: Show all suppliers with secret shares.
11. Detail 6: The user clicks 5 times on suppliers in 30 seconds.

12. Search 6: Show all suppliers with secret shares which won contracts with value over 1 billion CZK (about 40 million euro). User watches the map from 10 to 20 seconds.

13. Detail 7: Showing detail one of the suppliers founded in the previous step and reading details from 20 to 30 seconds.

14. Detail 8: The user clicks 5 times in 30 seconds on the submitters of the contracts which were won by the supplier from the previous step.

The scenario of a common user's behavior is divided between searching with various filters and obtaining information about the public contracts, suppliers or submitters. We also consider an inactivity when user reads provided information. This inactivity has random length which helps with distribution of the requests on the server during the time. This is important especially when we test more concurrent users sending requests on the server. Single passage through the scenario generates 34 requests on the server.

7.2 Results and interpretation

Below are tables with results of benchmarks against mySQL database, D2R's SPARQL endpoint and Sesame's (with in memory repository) SPARQL endpoint. The tables show times in milliseconds which represents the delay between sending request and receiving the response from the server. The benchmark simulated different amount of users sending request against the server according to the scenario defined in the previous section. We provide average, median and maximum values of response times.

We performed every test 5 times. This means that for each user was the scenario repeated 5 times. So for example, if we simulated 50 users, then the scenario was executed 250 times and 8,500 requests was sent on the server. If we simulated more users sending requests on the server, we delayed the start of their simulation by 5 seconds against the previous started user. This means that testing tool started to simulate each user every 5 seconds. These delays between users reflect more reality. If we did not do it then server would be requested by the totally same requests from all users in the same time. The requests are better distributed in time when users are started with delays. The times in the tables does not contain times, when is simulated user's inactivity (e.g. when user reads some information and does not send any requests to the server). Presented numbers reflect times between sending request and receiving response.

The table 7.1 shows the results of benchmark PHP scripts delivering data from mySQL database. We performed simulation of 5, 25 and 50 concurrent

users. Average, median and maximum of times are presented. Last row contains average, median and maximum of respective columns.

mySQL benchmark									
	5 users			**25 users**			**50 users**		
	Avg	Med	Max	Avg	Med	Max	Avg	Med	Max
Search 1	651	539	2132	605	543	1766	748	623	3501
Search 2	354	220	1599	272	215	2444	343	229	3068
Detail 1	39	19	1287	43	17	2839	73	20	2619
Detail 2	37	24	105	37	23	371	86	26	1944
Search 3	241	210	797	228	209	646	305	219	2649
Detail 3	20	18	149	32	17	1213	72	20	2361
Search 4	84	39	1032	45	38	222	106	40	1997
Detail 4	358	126	1347	362	123	3377	590	237	5028
Detail 5	42	28	150	49	23	907	99	27	2714
Search 5	86	76	140	92	77	234	138	79	1763
Detail 6	42	35	296	62	35	2581	103	39	2392
Search 6	87	84	169	86	82	197	119	85	601
Detail 7	54	45	170	67	44	1189	99	47	1351
Detail 8	298	90	1073	333	67	3144	492	139	5572
TOTAL	**171**	**50**	**2132**	**166**	**51**	**3377**	**256**	**86**	**5572**

Table 7.1: The results of the mySQL benchmark

We can summarize the results as follows:

- MySQL database does not have any performance problems even with 50 concurrent users.
- There is almost no difference between 5 as 25 users. The times are very similar.
- Maximum times during 50 concurrent users are significantly higher. With 50 concurrent users is higher probability of the execution more requests at the one moment. This causes more workload on server and longer execution times. However, average times and especially median (the most often value) of times are very low. So response of the server is still very good.
- Search 1 and 2 generate the most workload on database together with Detail 4 and 8. It is understandable with Search 1 and 2, because after their requests is returned several thousand records. But why Detail 4 and 8? The answer is simple. Detail 4 and 8 do not request only details about submitter, but also all contracts with subcontracts submitted by submitter. This generates additional database querying.

The table 7.2 contains results of the benchmark against Sesame's SPARQL endpoint. Sesame used memory as a repository. We performed simulation of 1, 5 and 25 concurrent users. This table is structured in the same way as the previous table.

Sesame's SPARQL endpoint benchmark									
	1 user			**5 users**			**25 users**		
	Avg	Med	Max	Avg	Med	Max	Avg	Med	Max
Search 1	9929	9459	11805	18892	17455	38100	73118	71424	145621
Search 2	7766	7822	8317	13371	12534	22383	99760	104101	150394
Detail 1	3297	3250	3496	3735	3249	12440	22718	8344	128510
Detail 2	2230	2211	2318	2425	2241	3549	3429	2513	10322
Search 3	3435	3375	3655	3890	3312	7194	16107	16949	31430
Detail 3	3304	3264	3682	3359	3245	5521	8250	6281	28919
Search 4	1343	1308	1480	1310	1298	1502	1482	1371	4824
Detail 4	4385	3661	14158	4210	3343	14883	4548	4041	14855
Detail 5	2246	2243	2305	2212	2204	2470	2202	2195	2492
Search 5	1202	1183	1280	1215	1169	1545	1175	1159	1397
Detail 6	2298	2273	2484	2285	2238	4784	2268	2247	2875
Search 6	9723	9544	11035	14581	13972	28146	75148	71173	125092
Detail 7	2325	2287	2424	2268	2250	2396	2358	2264	5172
Detail 8	8376	7954	11781	8371	7734	15315	14040	9444	98728
TOTAL	**4419**	**3275**	**14158**	**5866**	**3254**	**38100**	**23329**	**4904**	**150394**

Table 7.2: The results of the Sesame's SPARQL endpoint benchmark

At first glance it is clear that querying Sesame's SPARQL endpoint is orders of magnitude slower than querying mySQL database. There is huge gap in performance even though Sesame uses memory as repository. We can state that:

- Long response times with 25 concurrent preclude the use of the application.
- The most time takes searching. Times of the other requests are long in comparison with mySQL but still sufficient even with 25 concurrent users.

The application would be possible to use with 5 concurrent users. The problem is that in some cases are extremely long responses. The maximum of first search is almost 40 seconds. As we can see in times with 1 user, first search takes almost 10 seconds. But the delay between starting simulation of users is only 5 seconds and Search 1 and 2 have the longest responses. The result is that times of these searches are long, while requests in the middle of table have still reasonable responses even for 25 concurrent users. So we decided to do another benchmark. This benchmark will have 15 seconds delay between starting of simulation of users. The results are in table 7.3.

Sesame's SPARQL endpoint benchmark			
	5 users		
	Avg	Med	Max
Search 1	11920	9857	21339
Search 2	9936	8052	21496
Detail 1	3806	3374	9534
Detail 2	2480	2213	4788
Search 3	3589	3290	7854
Detail 3	3448	3245	8753
Search 4	1426	1316	2735
Detail 4	3859	3304	12632
Detail 5	2214	2205	2345
Search 5	1173	1169	1344
Detail 6	2244	2231	2495
Search 6	4112	3946	4356
Detail 7	2419	2271	4522
Detail 8	8582	7874	14666
TOTAL	**3264**	**21496**	**5730**

Table 7.3: The results of Sesame's SPARQL endpoint benchmark

We can see that the longest times are considerably lower. The most important for an application usage with more concurrent users is distribution of more complicated searchings. The application using Sesame's SPARQL endpoint can handle only several users if they perform complicate searchings. But if they perform less complicated actions which lead to less complicated SPARQL queries then the application can handle up to 25 users.

Now should follow the table with the results of D2R's SPARQL endpoint benchmark. As we discussed in 5.5 D2R has problems even with the simple queries. Especially when we query data about the public contracts. We were not able to finish our benchmark from this reason. The first search from our scenario caused maximum processor workload and results were not returned even after several minutes. We had to manually kill mySQL database process (D2R server maps data from mySQL database). So there was no sense to continue with the benchmark.

At the end we show figure 7.1 with the progress of average and medium times of requests during simulation of 5 users against mySQL database. We can clearly see that after some time the response of the server remains constant and stable. Initial 'waves' in the times can be easily explained. We try to distribute requests on the server in the time, but anyway at the beginning are requests sent in the same time and users are inactive also in the same time. This means that more or less demanding request are executed in the same time which leads to the 'waves'

in graph. Server's response times are getting stable as users' requests get more distributed in the time by a random waiting times defined in the scenario of user's behavior. This pattern repeated in every benchmark. This proves that our benchmark was properly designed because users' requests were uniformly distributed in the time.

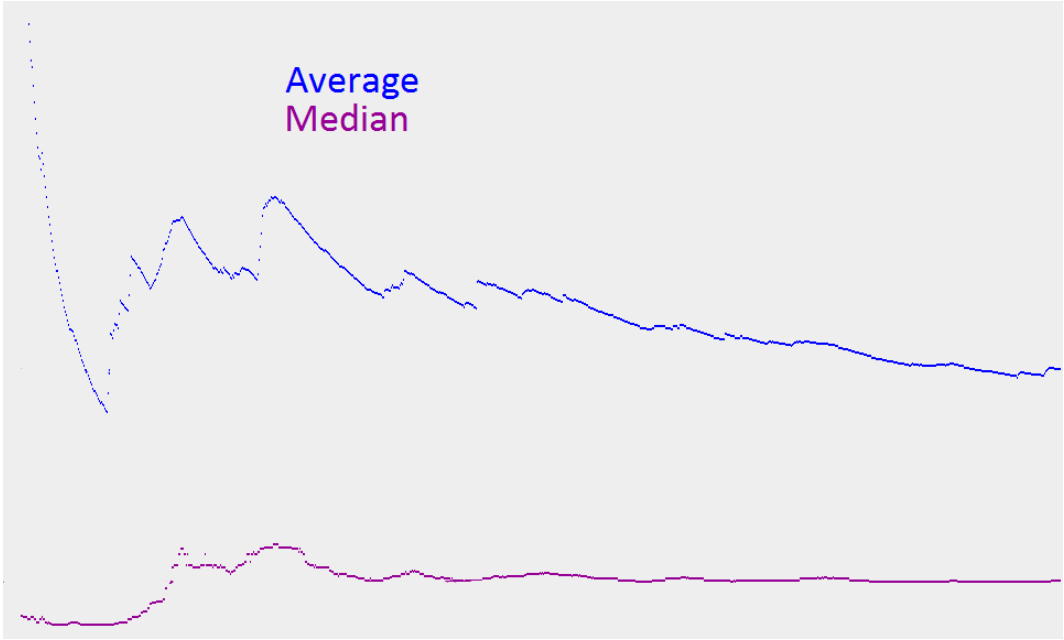


Figure 7.1: 5 users and mySQL

7.3 Conclusion

The conclusion of this chapter is that there is still a huge performance gap between the traditional relational databases (mySQL), triple stores (Sesame) and mapping tools (D2R). D2R's SPARQL endpoint was not able to perform our queries in a reasonable time. Sesame's SPARQL endpoint was able to handle only several users at one moment while mySQL database was able to handle 50 users with no performance problems. We must state, that the performance of tested SPARQL endpoints is not sufficient for applications serving many users.

Chapter 8

Conclusions

8.1 Linked Data basics

This section will give a brief manual for the beginning developer. At the beginning you need to decide between two basic scenarios:

- **Scenario 1:** Linked Data as primary data source. You will create your data directly as RDF and you will use SPARQL endpoint as data source. You should keep on mind the performance problems of the current triples stores and SPARQL endpoints. Some tools have also problems with diacritics. Please see chapter 7.
- **Scenario 2:** Publishing existing (usually relational) dataset as Linked Data. You can still use the relational database as a data source in case of some performance problems of SPARQL. The problem is that is complicated to enrich your dataset with the links from the interlinking process.

Now you need to do following steps:

1. Choose the ontologies fitting your data. Well-know vocabularies (FOAF, SIOC, SKOS, vCard, Dublin Core) should be reused wherever it is possible. New terms should be defined only if they are not contained in existing vocabularies.
2. Choose the triple store (first scenario) or the mapping tool (second scenario). The tools helping with Linked Data transformation are discussed in chapter 4. D2R server is most often tool used for mapping relational database to RDF.
3. Perform interlinking. Enriching your dataset with the links from interlinking process can be problem if you have chosen the second scenario. Mapping

tools only maps data from the relational database. There is no possibility to add the links to your database. Interlinking tools has also some limitations as is discussed in section 5.6.

4. Publish your data in HTML and RDF. Mapping tools are usually able to publish your data. But this is not the case of all triple stores. However, there exist tools which can use triple stores as data source and publish their data. These tools are also discussed in chapter 4.

If you are going to create an application consuming Linked Data you can use public SPARQL endpoints to enrich your application. Our experiences from enriching application on the fly from DBpedia are discussed in section 6.4. Keep on mind SPARQL performance, possible problems with diacritics and problems that names of the RDF properties are usually only in English. So for example, if you want to show additional data from DBpedia in *property:value* style you need to manually translate the names of properties.

8.2 Contributions and future work

The main goal of this thesis was to help developers which decided to experiment with Linked Data. Let's repeat what we have done so far:

- At the beginning we described the Semantic Web. We showed the Semantic Web technology stack and explained the most important technologies. We also explained what is relationship between Semantic Web and Linked Data.
- We explained Linked Data, its principles and steps for Linked Data creation. An overview of existing Linked Data sets and applications were given. We explained in detail the most important technologies (RDF, ontologies, SPARQL).
- We provided an overview of tools for all steps of Linked Data creation - RDF converters, triple stores, tools for mapping relational data to RDF, tools for automatic interlinking process and tools for publishing Linked Data on the Web. We also provided sources where potential readers can find more information.
- We presented our experiences from Linked Data creation and difficulties which can be expected during this process. We discussed obtaining data (scraping from web pages), their mapping to RDF, extending Public procurement vocabulary and problems we had with D2R mapping tool or with

Sesame triple store. We described in detail interlinking our dataset with other datasets and difficulties which are not clear and unexpected at the beginning.

- We develop the application consuming data from ordinary data source (mySQL) and SPARQL and compared both approaches. We also discussed difficulties we had during enrichment our application by data from DBpedia in on the fly manner.

Going back to our research goal we can state that the goal has been achieved. We gave an overview of tools involved in all steps of transformation regular dataset to Linked Data. We presented our experiences we had from creating our Linked Data set and developing of the application consuming it. Important is that we described in detail all difficulties we had during creating Linked Data and application consuming them. A lot of discussed difficulties are unexpected for the developers beginning with Linked Data. Potential readers of this thesis can take them into account. They do not have to spend hours by testing different tools and they can immediately choose the tools which suit the best their needs. Moreover, with the knowledge we acquired and presented in this thesis, developers can more easily decide if using Linked Data will be beneficial for them.

Our future work will be mainly focused on downloading/scraping more governmental data, their transformation to Linked Data and integration/interlikning with data we already have. This heavily depends on the amount of data which will be published by government. We believe that our application can help with pressure on government to publish more data in machine readable form. This will support further development of similar kinds of applications. As an example of data source which currently offers information only in human readable form can be web page with results of votes[1]. This data can be used for revealing conflicts of interests. A person which decides about the winner of public contracts is a member of a private company bidding for public contract.

Our data are accessible via public SPARQL endpoint and they are also accessible by traditional web browsers as every public contract, supplier or submitter has its own unique URL. Thanks to our effort are potential consumers of governmental data shielded from scraping web pages or learning new APIs. They only need to learn SPARQL query language and get familiar with Public procurement vocabulary and then they can access data gathered from various sources.

[1] www.volby.cz

8.3 Issues and research challenges

SPARQL and triple stores. From our point of view is querying with SPARQL query language very easy and simple. SPARQL implicit join syntax, possibility to query any data source with an unpredictable and unreliable structure makes from SPARQL very powerful query language for Linked Data. The biggest benefit of SPARQL is that once we already get familiar with some ontology we can query any SPARQL endpoint which use it. During writing SPARQL queries we missed analogy of Intellisense feature from Microsoft Visual Studio or from another developing environments. This feature automatically finishes code or gives hints. Let's consider following SPARQL query:

```
PREFIX pc: <http://purl.org/procurement#>
SELECT ?subj WHERE {
    ?subj rdf:type pc:Contract.
}
```

When we wrote this query, we had to know that the class Contract exists in the public procurement vocabulary. The feature which we propose would offer us automatically all classes from public procurement vocabulary after prefix *pc:* is typed. It would be very helpful for everybody who is querying new SPARQL endpoint and does not know some of ontologies used in dataset.

There exist already many triple stores. Some of them use traditional relational database for storing RDF triples, some of them use their own solution for storing RDF triples. There is also cloud based solution - the Talis Platform. However, they do not have achieve performance as existing relational database engines. Moreover, a lot of them does not support SPARQL insert so their SPARQL endpoints can be used only for data reading.

Ontologies. A lot of ontologies was already published. They are well known, settled and stable. For example, ontologies describing relationships between people (FOAF), Dublin Core ontology or GoodRelations Ontology for describing products, their features and prices. We found out several issues which should be solved. First was how to find ontologies we need. Ontologies should be reused wherever it is possible, but to find some which are not well known could be a problem which leads to creation of duplicated ontologies. We found second issue during implementation of enrichment our application with data from DBpedia. We wanted to show information in format *property: value*. The problem is that properties names are usually only in English and in language of creator of ontology. Last issue we realized when we updated public procurement ontology. Users of ontologies must check manually definition of ontology in order to know

if ontology is updated. Web of Data is evolving very fast same as ontologies and there is danger that a lot of data could be published in wrong format because used ontologies can be updated very often.

Mapping relational data to RDF. One of the biggest challenges within the Semantic Web is the conversion of relational databases to RDF. There exist several tools such as D2R server or Triplify. The problem is that it is hard to generate ontologies automatically from a dataset or to use automatically existing ontologies. Manual work is necessary to achieve a nice mapping. Furthermore, every tool proposes different approach for mapping or introduces different language for mapping. It is very time consuming to learn always new approach while searching for appropriate tools which pass the best our needs.

Interlinking. There exist already a lot of tools for interlinking process. They differs by domain in which they can be used (e.g. publications) or type of data they work with (e.g. multimedia data). Unfortunately it is still necessary to check their output manually, especially output from domain independent tools as Silk. Another problem is that each tool has different approach for definition how links between sources should be created. So potential user searching for appropriate tool need a lot of time to learn it.

Currently there is no widely accepted approach for handling broken and invalid links even though there exist solutions like DSNotify or Web of Data - Link Maintenance Protocol.

8.4 Criticism

For completeness we should mention that there exist also negative opinions on the Semantic Web and Linked Data. Some even call Semantic metadata as 'metacrap'. Metacrap is a portmanteau drawn from metadata and crap. Criticism mostly mentions:

- People lie. People could use wrong or incorrect metadata in order to be (for example) on better positions in semantic search engines.
- People are lazy. People use metadata in incorrect way or metadata could be incomplete which can lead to confusion of Semantic web indexers.
- Censorship and privacy. This is probably the biggest issue. "*Enthusiasm about the semantic web could be tempered by concerns regarding censorship and privacy. For instance, text-analyzing techniques can now be easily bypassed by using other words, metaphors for instance, or by using images*

in place of words. An advanced implementation of the semantic web would make it much easier for governments to control the viewing and creation of online information, as this information would be much easier for an automated content-blocking machine to understand. In addition, the issue has also been raised that, with the use of FOAF files and geo location metadata, there would be very little anonymity associated with the authorship of articles on things such as a personal blog." [26].

8.5 Conclusion

Linked Data helped to overcome the Chicken and egg problem of the Semantic web. The Linked Data cloud grows and this growth is accelerating same as the amount of tools or applications for Linked Data consumption, transformation, storing, interlinking or mapping from relational databases. This growth and research of new tools is also supported by the Linking Open Data 2 project co-funded by the European Commission. Authors and contributors of idea of Semantic web believed that transformation of current web into Semantic web will be fast and rapid. Now we see that this did not happen and that it is not even possible because a lot of issues are still not solved. Author believes that Linked Data and applications based on them will grow and Linked Data will become to be a mainstream. But this transformation of current web will be smooth and continuous process.

Appendix A

The content of the attached CD

The attached CD contains:

- an electronic version of the thesis
- the web crawler with source codes
- the mySQL database dump with scraped data
- the extended Public Contracts Ontology
- D2R server with mapping file
- interlinking files (Silk)
- the benchmark definition files
- the documentation and the manual for installation of the created application

Please see *README.txt* file on the root of the CD. The file contains description of the folder structure and the CD's content.

Bibliography

[1] Internet Growth Rates: *http://www.livinginternet.com/i/ip_growth.htm*, 2010

[2] Michael Hausenblas: *Exploiting Linked Data For Building Web Applications* , 2009

[3] Berners-Lee, Tim; James Hendler and Ora Lassila: *The Semantic Web*, Scientific American Magazine, 2001

[4] W3C Linked Data: *http://www.w3.org/standards/semanticweb/data*, 2010

[5] Berners-Lee,Bizer,Heath: *Linked Data: The Story So Far*, 2009

[6] Linked Data Design Issues: *http://www.w3.org/DesignIssues/LinkedData.html*, 2010

[7] D. Ayers. Evolving the Link. IEEE Internet Computing, 11(3):94-96, 2007

[8] Datasets in LOD Cloud: *http://www4.wiwiss.fu-berlin.de/lodcloud/*, 2010

[9] Michael Hausenblas: *Linked Data Applications - The Genesis and the Challenges of Using Linked Data on the Web*, 2009

[10] LOD2 : *http://lod2.eu/* , 2011

[11] Brice Dunwoodie - What is an Ontology? And Why We Need Them: *http://www.cmswire.com/cms/knowledge-management/what-is-an-ontology-and-why-we-need-them-001479.php* , 2011

[12] Natalya F. Noy, Deborah L. McGuinness - Ontology developement 101: A guide to creating your first ontology: *http://protege.stanford.edu/publications/ontology_development/ontology101-noy-mcguinness.html* , 2011

[13] Lee Feigenbaum: *http://www.thefigtrees.net/lee/sw/sparql-faq#benefits* , 2011

[14] Bizer, Lehmann, Kobilarov, Auer, Beckera, Cyganiakc, Hellmannb: *DBpedia-Acrystallization point for theWeb of Data* , 2009

[15] DBpedia: *http://www.dbpedia.org/*, 2011

[16] Wikipedia: *http://www.wikipedia.org/*, 2011

[17] Christian Bizer, Andreas Schultz: *Benchmarking the Performance of Storage Systems that expose SPARQL Endpoints* , 2008

[18] Bin He, Mitesh Patel, Zhen Zhang, Kevin Chen-Chuan Chang: *Accessing the Deep Web: A Survey* , 2007

[19] Sören Auer, Sebastian Dietzold, Jens Lehmann, Sebastian Hellmann, David Aumüller: *Triplify - light-weight linked data publication from relational databases.* , 2009

[20] Cerbah, F.: *Learning Highly Structured Semantic Repositories from Relational Databases - RDBtoOnto Tool* , 2008

[21] Satya S. Sahoo, Wolfgang Halb, Sebastian Hellmann, Kingsley Idehen, Ted Thibodeau Jr, Sören Auer, Juan Sequeda, Ahmed Ezzat: *A Survey of Current Approaches for Mapping of Relational Databases to RDF* , 2009

[22] Stephan Wölger, Katharina Siorpaes, Tobias Bürger, Elena Simperl, Stefan Thaler, Christian Hofer: *Interlinking data - approaches and tools*, 2011

[23] Julius Volz, Chris Bizer, Martin Gaedke, Georgi Kobilarov: *Discovering and Maintaining Links on the Web of Data* , 2009

[24] Oktie Hassanzadeh, Reynold Xin, Renée J. Miller, Anastasios Kementsietsidis, Lipyeow Lim, Min Wang: *Linkage Query Writer* , 2009

[25] Haslhofer B. , Popitsch, N.: *DSNotify - Detecting and Fixing Broken Links in Linked Data Sets* , 2009

[26] Wikipedia - Semantic Web: *http://en.wikipedia.org/wiki/Semantic_ Web#Skeptical_ reactions/*, 2011